FREE OS/2 MAGAZINE!

A GUIDE TO DEVELOPING ON

OS/2®

A SUPPLEMENT TO COMPUTER LANGUAGE

TIPS FOR OS/2 SUCCESS!

C++ for OS/2 from Borland and IBM

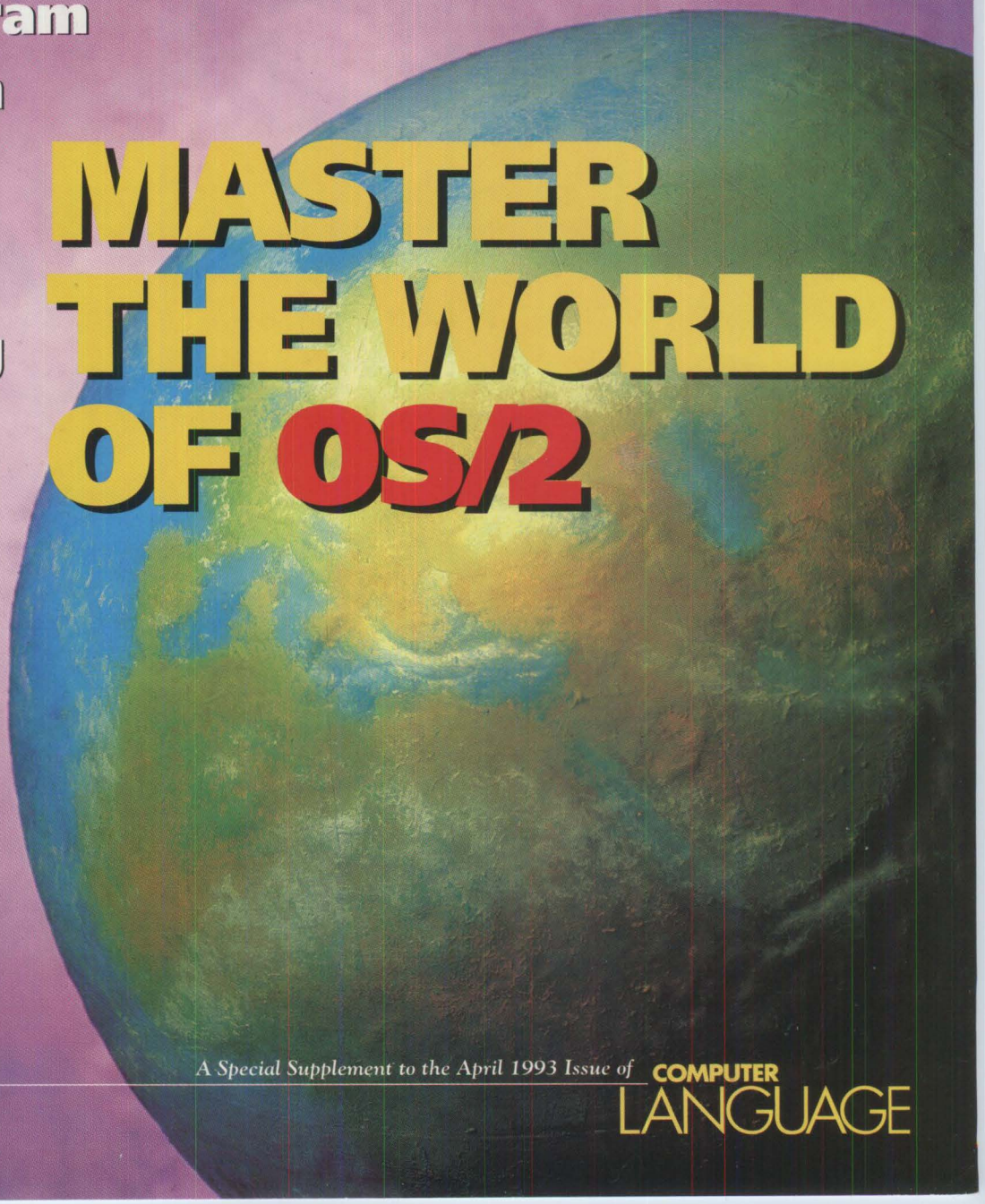
How to Program OS/2's System Object Model

Programming Containers

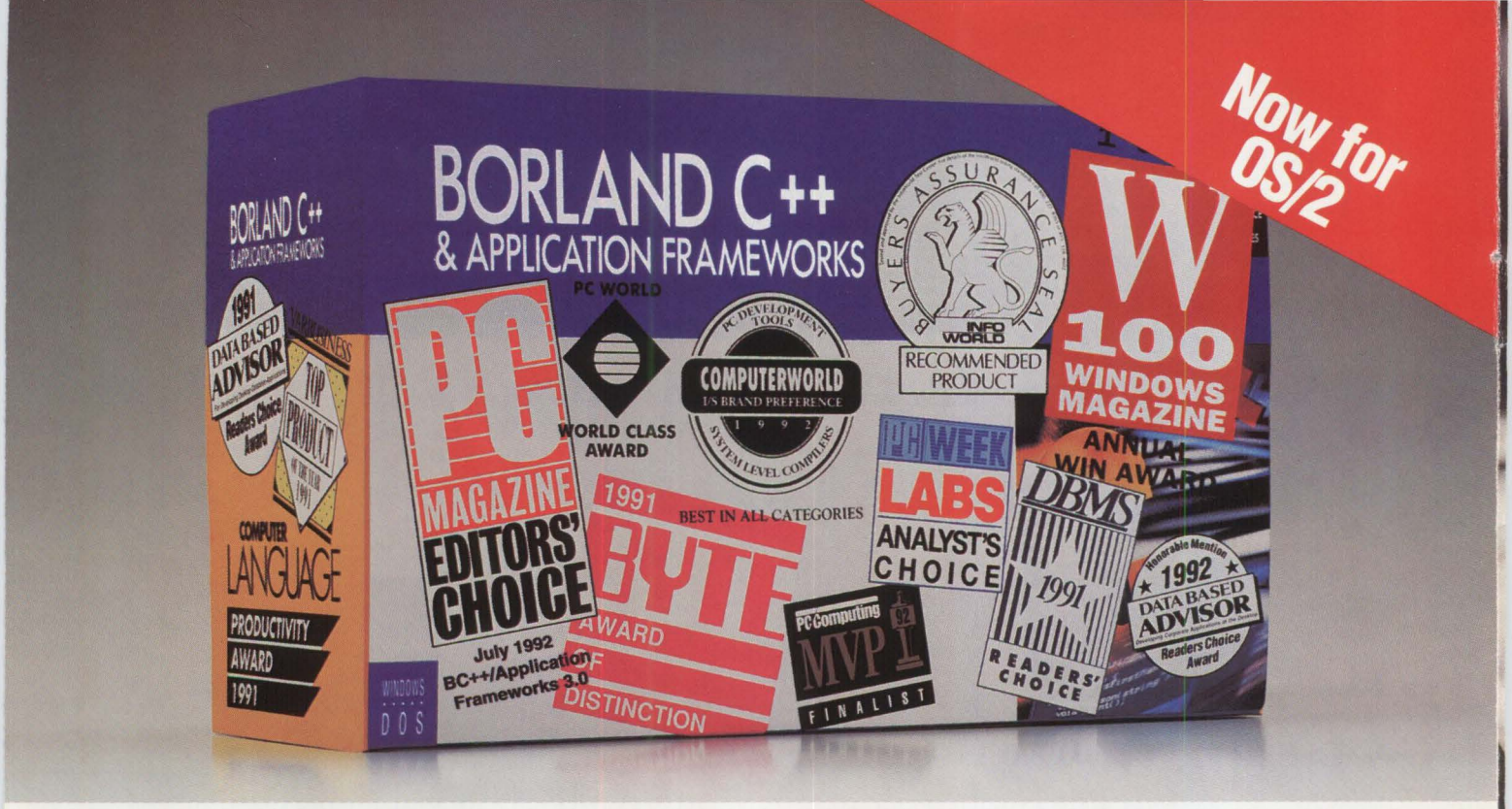
Building Notebooks

David Moskowitz: Persistence Pays Off

MASTER THE WORLD OF OS/2



A Special Supplement to the April 1993 Issue of COMPUTER LANGUAGE



Undefeated champion!

Borland C++ leaves pretenders playing catch-up.

REPORT CARD AUGUST 17, 1992		INFO WORLD
C++ Windows development	Borland C++ & Application Frameworks version 3.1	Microsoft® C/C++ version 7.0
Performance		
Programming environment	Excellent	Good
Language	Very good	Good
Productivity	Excellent	Good
Documentation	Very good	Excellent
Ease of learning	Very good	Good
Ease of use	Excellent	Good
Support		
Support policies	Excellent	Very good
Technical support	Very good	Very good
Value	Excellent	Very good
Final scores	**8.9**	**6.9**

Borland C++ consistently outscores Microsoft C/C++ in performance, ease of use, and value.

Borland C++ & Application Frameworks™ is unquestionably the best C and C++ compiler and tools for building applications. That's why it is the clear winner in every review and product comparison. And why more than one million professional programmers rely on Borland C++.

Why is Borland C++ the best? Because its highly visual tools and Integrated Development Environment make it easy to create high-performance Windows applications. And since Borland C++ is the only popular compiler that meets the certified* ANSI C and AT&T C++ standards, you know your investment in Borland C++ will take you well into the future.

Now in its third generation, Borland C++ gives you the features and reliability you can trust for DOS, Windows, and now OS/2.®

1992 BYTE AWARD OF EXCELLENCE

1992 – WINNER – TECHNICAL EXCELLENCE PC MAGAZINE
DEVELOPMENT TOOLS
Borland C++ 3.1
Borland International, Inc.

PC WORLD
WORLD CLASS AWARD 1992

Go with Borland C++, the #1-selling C and C++ application development system.

See your dealer or call now, 1-800-331-0877, ext. 5040

In Canada, call 1-800-461-3327.

90-DAY GUARANTEE MONEY-BACK

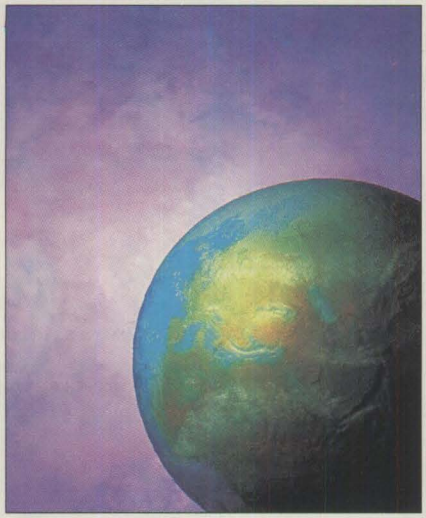
Borland® C++
for Windows, DOS, and OS/2

*ANSI C certification awarded by the British Standards Institute.

Circle No. 801 on Inquiry Card

CONTENTS

APRIL 1993



The world of OS/2

On the cover: Join us in our in-depth coverage of the world of OS/2 2.0!

Carter Dow Photography

Articles

Product wrap-up

Editorial

ResourceShield

The Only Tool You'll Ever Need

... to create, modify, manage, browse and translate resources such as dialog boxes, bitmaps, icons, cursors and menus.

Increase the productivity of your development work by organizing resources into projects and seemlessly integrate all the resource editors.

Reduce the amount of time required to design the user interface of your application—use the state of the art visual editors to interactively design and modify dialogs, bitmaps, icons, cursors, menus, accelerators, string tables and all other resources—or just use any of the hundreds of professionally designed resources included.

Capture images from anywhere on the screen directly into the image editor. Create bitmaps, icons and cursors from the captured image.

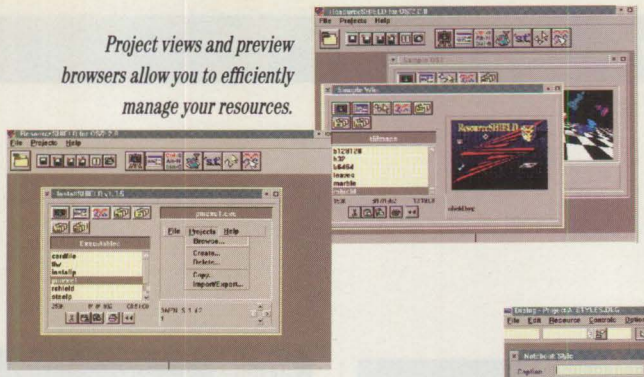
Customize the look and feel of your applications—directly extract, modify or replace any resource in any program file, EXE, DLL—no source code is needed.

Translate applications into foreign languages without having access to the applications source code—directly modify all messages, menus and dialogs into any national language.

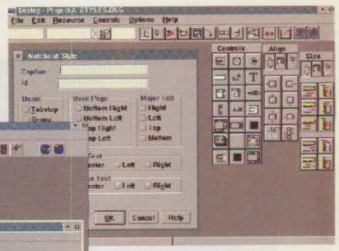
Edit, copy and paste resources directly into and from any EXE, RES, DLL, RC program file. Eliminate the time consuming edit-compile-link cycle—increasing your productivity dramatically.

Migrate your applications to new 32-bit OS/2 2.0 and NT applications utilizing the transparent and automatic resource translation capabilities. Create your resources once on any platform and instantly use them on any other platform.

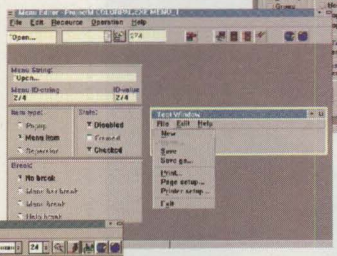
Project views and preview browsers allow you to efficiently manage your resources.



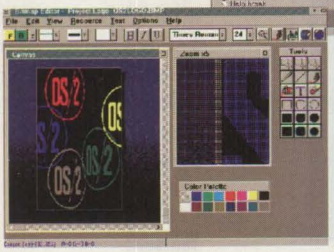
Quickly design complex dialogs using powerful design tools in the Dialog Editor.



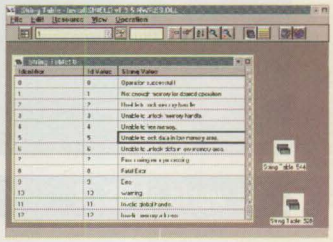
The Menu Editor.

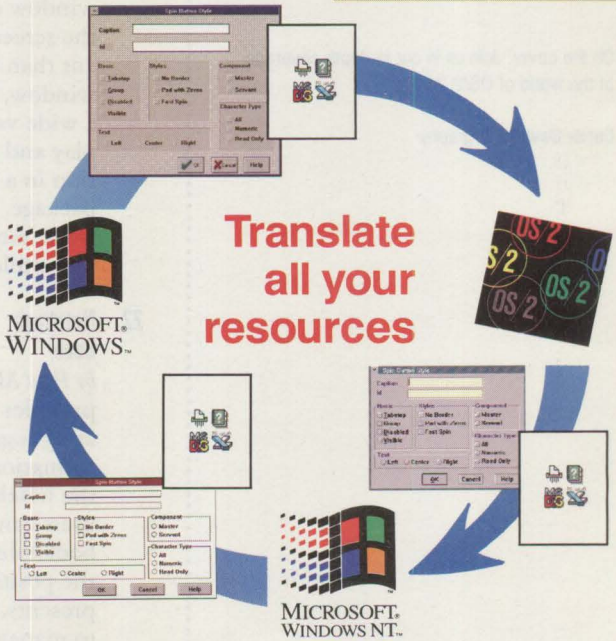


Create bitmaps, icons, cursors using the advanced image editors.



Find, replace and translate text messages in string tables.





The Stirling Group

Making it easy to make™

172 OLD MILL DRIVE • SCHAUMBURG, IL 60193 • USA **1-800-3 SHIELD**

1-800-374-4353

CALL 708-307-9197 • FAX 708-307-9340

CompuServe: GO STIRLING • MCI Mail: *STIRLING*

Circle No. 802 on Inquiry Card

COMPUTER LANGUAGE

EDITOR:
Larry O'Brien
SENIOR EDITOR
Nicole Freeman
SUPPLEMENT EDITOR
David Moskowitz
PRODUCT REVIEW EDITOR:
Thomas Murphy
DEPARTMENTS EDITOR:
Michelle Gahee
EDITORIAL ASSISTANTS:
Michelle Williams and Gen Rowland
CONTRIBUTORS:
Roger Sessions, Nurcan Coskun, Paul Montgomery, Randy Branham, and Chris Corry

PUBLISHER:
Regina Starr Ridley
ASSOCIATE PUBLISHER:
Veronica Costanza
GROUP DIRECTOR:
Donald Pazour
NATIONAL SALES MANAGER/ ADVERTISING SALES STAFF:
Veronica Costanza (212-626-2418)
David Moreau (212-626-2318)
(Middle Atlantic/South)
REGIONAL SALES MANAGERS/ ADVERTISING SALES STAFF:
Jo Ben-Atar (212-626-2351)
Michele Blake (212-626-2322)
(New England/Midwest)
Cathy Passage (415-905-2392)
Pam D. Grossman (415-905-2256)
Michele Anet (415-905-2298)
(West/Southwest)
MARKETING MANAGER:
Susan McDonald
ART DIRECTOR/MARKETING:
Christopher H. Clarke
ADVERTISING PRODUCTION COORDINATOR:
Charles Hennen
DIRECTOR OF PRODUCTION:
Andrew A. Mickus
DIRECTOR OF CIRCULATION:
Jerry M. Okabe
CIRCULATION MARKETING MANAGER:
Gina Oh
NEWSSTAND COORDINATOR:
Pam Santoro
REPRINTS:
Andrea Varni
(415) 905-2200

FOUNDING PUBLISHER:
Carl Landau

ABP

Persistence pays off

It started in 1987 when IBM and Microsoft announced the OS/2 operating system. I won't bore you with the details. It is no secret that OS/2 1.x was not a stunning success. It did attract a following—a group of developers and users who saw and understood the promise.

OS/2 1.0 was missing the Presentation Manager. OS/2 1.1 shipped with Presentation Manager but we still couldn't print. OS/2 1.2 added the high-performance file system. OS/2 1.3 was finally small and agile enough to run on non-power-user systems. Fade out, fade in, and we get to OS/2 2.0—it could print, it had the Workplace Shell, it had promise! OS/2 2.1 adds more support for SCSI devices and CD-ROM (OS/2 2.1 is even distributed on CD-ROM). It has a fast 32-bit graphics engine, too.

But wait—there's more! A year or two ago, you wouldn't expect to read very much in some of the rumor columns about a small company developing an OS/2 product; it wasn't news—then. Yet, when something appeared recently, the phone at UCANDU Software Inc. in Carey, NC, started to ring. The company had almost 100 calls in two days. It seems a lot of people are interested in its Visual Programming with REXX tool—and the rumor said the product wouldn't be ready until April 1993.

Very often, it is the small, unknown companies that create the products that make the platform. Who had ever heard of Aldus before the Macintosh, Micrografix before Windows, or Microsoft before IBM. UCANDU Software is convinced.

For years, we've told anyone who would listen that small developers' companies were the future and hope for OS/2. We urged small developers to persist with their OS/2 development because we knew it would pay off. Many companies attended workshops we ran for IBM to help developers migrate their applications to OS/2 1.x and 2.0. Some of them were ready to beta test their 16-bit product by the end of the workshop week. Yet, their marketing departments decided there was a limited future. Today, other companies have filled the void; companies who persisted.

Small developers aren't alone; the roll call of large corporations using OS/2 2.1 in strategic ways is

COMPUTER LANGUAGE *(ISSN 0749-2839) is published monthly by Miller Freeman Inc., 600 Harrison St., San Francisco, CA 94107, (415) 905-2200. Please direct advertising and editorial inquiries to this address. Subscription rates for the U.S. are $34.95 for 12 issues, $44.95 for 24 issues, and $54.95 for 36 issues. Canadian/Mexican orders must be accompanied by payment in U.S. funds with additional postage of $6 per year. All other foreign must be prepaid in U.S. funds with additional postage at $15 per year for surface mail and $40 per year for air mail.* ***POSTMASTER:*** *Send address changes to* COMPUTER LANGUAGE, *P.O. Box 53525, Boulder, CO 80322-3525. Send subscription orders and inquiries to P.O. Box 53525, Boulder, CO 80322-3525. For customer service, telephone toll-free 1-800-288-1322 (in Colorado, (303) 447-9330). Second class postage is paid at San Francisco, Calif., and at additional mailing offices. Please allow six weeks for change of address to take effect.* COMPUTER LANGUAGE *is a registered trademark of Miller Freeman Inc., a member of the United Newspapers Group.*

rapidly increasing. With the large numbers comes opportunities and demand for software. Wordperfect, Lotus, Borland, Symantec—all the major players have 32-bit OS/2 applications in the works.

IBM continues to upgrade the system to make it better. The first release included the Workplace shell, an object-oriented user interface. Buried under the covers was a technology called SOM (the System Object Model). SOM provides a way to describe objects that is independent of a programming language. Roger Sessions discusses SOM in "SOM Enchanted Evening...," starting on p. 7.

Any new system creates new issues for developers, in this case some new controls. It's possible to write whole volumes on the Container and Notebook controls. Randy Branham and Paul Montgomery provide an excellent introduction to these devices in "Container Yourself," starting on p. 16, and "Bene Notebook," starting on p. 22. Of course, no new operating system would be complete without a programming language. So, we asked Chris Cory to provide a review and comparison of both Borland International's C++ for OS/2 and IBM Corp.'s C Set/2 v. 2.0, starting on p. 27.

As I said, it started in 1987. Six years later we see evidence that persistence really does pay off. IBM listened; the result is OS/2 2.1. It is here, and it works. *COMPUTER LANGUAGE* editor Larry O'Brien and I talked about this supplement at Fall COMDEX. As a result, I requested contributions from a number of people I've had the good fortune to get to know. At deadline, we had more articles than space in this issue allows. The folks at *COMPUTER LANGUAGE* had to make some hard choices; the results can be found on the following pages. To everyone who worked with me to help put this together, "Don't give up, persistence pays off!"

David Moskowitz,
Supplement Editor

COMPUTER LANGUAGE has its own forum on CompuServe! If you're interested in talking with our editors, authors and columnists, and industry gurus, or you just want to download program files, you'll want to join CLMFORUM. For a free introductory membership, call (800) 848-8199 and ask for representative 129. You'll receive a personal ID number, a $15 introductory usage credit to explore CLMFORUM and CompuServe's other offerings, and an introductory subscription to *CompuServe Magazine*.

Miller Freeman
A MEMBER OF THE UNITED NEWSPAPERS GROUP

un A United Newspapers publication

COMPUTER LANGUAGE CODE

COMPUTER LANGUAGE code can be obtained from CompuServe (type "GO CLMFORUM" and you'll be in our forum) and bulletin boards in the U.S. and Canada.

BBS parameters are eight data bits, no parity, one stop bit, full duplex, and 300 or 1200 baud.

CompuServe
GO CLMFORUM

Seattle, Wash.
(206) 848-9232

New York, N.Y.
(212) 980-0770

Sunnyvale, Calif.
(408) 969-4159

East Lansing, Mich.
(517) 353-2003

Ottawa, Ont., Canada
(613) 738-1793

Worthington, Ohio
(614) 885-9829

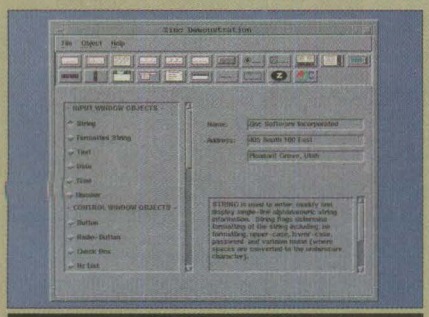
Eureka, Calif.
(707) 444-9203

Woodstock, Ill.
(815) 337-0279

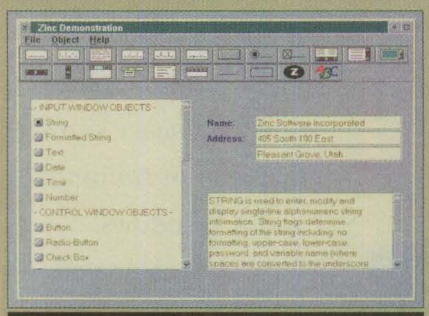
Sydney, Australia
(02) 449-9821

COMPUTER LANGUAGE code is available in Austria from the IBM PC User Group: Hasibederstraße 2/64/6, P.O. Box 40, A-1225, Vienna, Austria, fax: (0222) 220-6096.

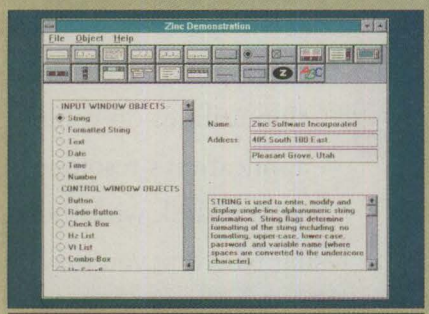
THINK ZINC



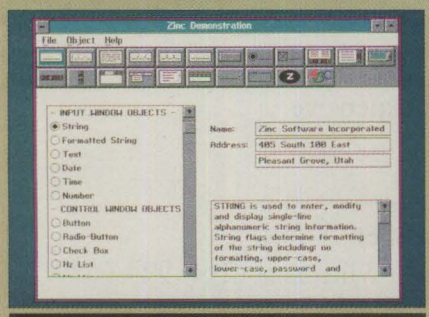
UNIX MOTIF



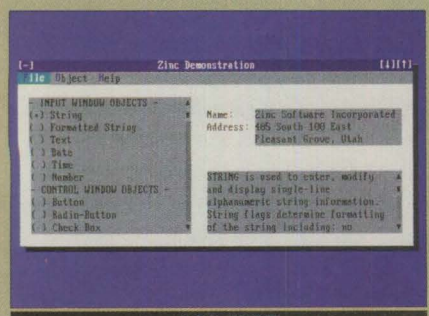
OS / 2 2.0



MS WINDOWS & WINDOWS NT



DOS GRAPHICS



DOS TEXT

ZINC™ APPLICATION FRAMEWORK™ 3.5

Multiplatform Flexibility From One Application Framework.

FLEXIBILITY *is an essential component in software design. Your development tools should allow you to easily incorporate new features and technologies into your applications without rewriting all of your code. That's a tall order if your development tools are designed like a straight-jacket. Zinc Application Framework 3.5 and Zinc Designer™ give you flexible and extendible support for Microsoft Windows, Windows NT, OS/2 2.0, UNIX Motif, DOS Graphics and DOS Text. And Zinc does it with ONE set of source code. Zinc's multiplatform, object-oriented architecture won't confine your application development options today... or tomorrow.*

FOR A FREE ZINC DEMO KIT, CALL US TOLL FREE TODAY AT

1.800.638.8665. IN EUROPE CALL +44 (0)81 855 9918

z i n c

ZINC SOFTWARE INCORPORATED, 405 SOUTH 100 EAST, 2ND FLOOR, PLEASANT GROVE, UTAH 84062, TEL 801.785.8900, FAX 801.785.8996, BBS 801.785.8997.

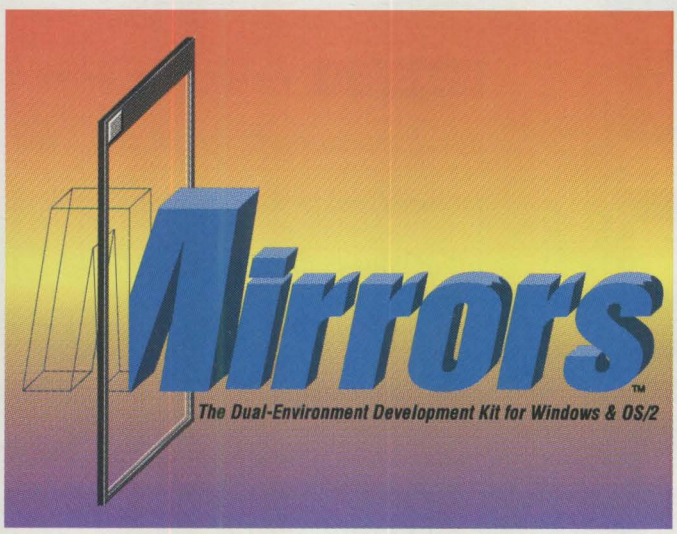
EUROPE: ZINC SOFTWARE (UK) LIMITED 58-60 BERESFORD STREET, LONDON SE18 6BG, TEL +44 (0) 81 855 9918, FAX +44 (0) 81 316 7778, BBS +44 (0) 81 317 2310

Circle No. 803 on Inquiry Card

Changing Windows™ Applications Into OS/2® Applications Is

A Neat Trick Done With Mirrors

Micrografx Mirrors™ can help you work magic. You can reach over a million new users for the price of an inexpensive tool kit. With Mirrors, you can maintain an application for two operating systems with a single set of source. This means you aren't forced to choose between operating systems, and you won't get bogged down in version control problems and divided development efforts.



Get the advantages of OS/2: great performance, access to a true multi-tasking operating system, 32-bit architecture, a more stable platform, integration with other OS/2 applications, and a clear migration path — not to mention the advantage of a million new customers.

How does it work? It's simple. Mirrors emulates Windows. When your application, running under OS/2, calls a Windows function, Mirrors intercepts the call. Mirrors then implements it using functions within the OS/2 system DLLs. Mirrors transforms data returned by OS/2 and passes it back in a form that Windows applications understand. Your application may never know that it's not running under Windows.

What do you need to do to make this happen? First, run Micrografx's conversion utilities on your application's resources, then re-link with the Mirrors DLL. That's it. Using Mirrors, you may not even need to recompile. Micrografx developed Mirrors. That means this tool kit was written by Windows developers for Windows developers. Mirrors is fast and inexpensive. Look into it!

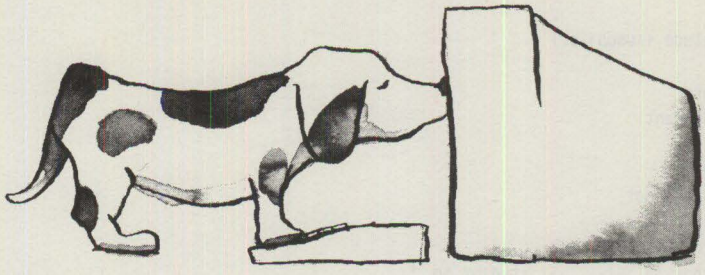
To purchase your copy of Mirrors today, call (214) 994-6566.
To learn more about how this trick is done, call Micrografx Technical Support for Mirrors at (214) 994-6659.

· Mirrors is a 32-bit DLL for increased performance · 32-bit Mirrors DLL provides support for 16-bit applications
· Non-debug and debug versions of Mirrors DLL provide handle validation and error reporting
· Automated conversion of Help, bitmaps, cursors, and icons · Includes DOS and OS/2 host independent file I/O libraries
· Interrupt 21 directly supported with no need to modify ASM files · DOS3CALL interrupt support
· Dynamic Data Exchange support with native PM applications · Mirrors also supports Clipboard data sharing

MICROGRAFX®

Circle No. 804 on Inquiry Card

SOM Enchanted Evening...



Using OS/2 2.0's System Object Model will enhance existing object-oriented languages and bring object orientation to your procedural projects.

by Roger Sessions and Nurcan Coskun

Object-oriented programming is revolutionizing the way we write software. In object-oriented programming, we employ three major features: encapsulation, to write multipurpose and easily tested classes; inheritance, to improve code reuse; and polymorphism, to simplify code and reduce maintenance costs.[1]

SOM is an important technology designed to significantly enhance existing object-oriented programming languages and bring object-oriented capabilities to languages that are essentially procedural.[2,3]

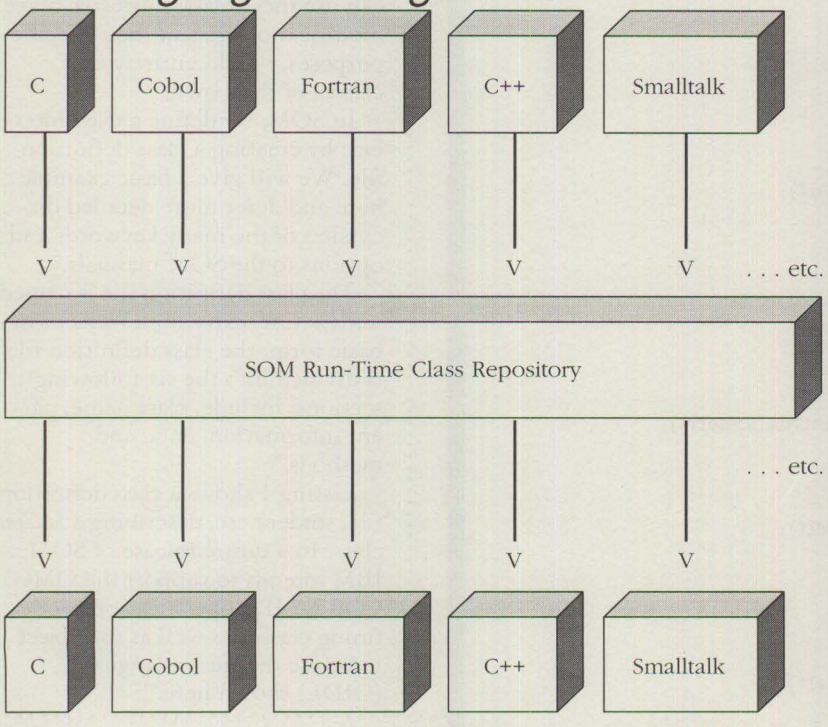
The most important feature of SOM and the one that most sets it apart from other object-oriented technologies is its language neutrality. SOM is designed to allow classes written in one language to be used from any other language. When we say these classes can be used, we mean this in the fullest possible object-oriented sense. The classes can be instantiated, manipulated, and, most important, subclassed. The only requirement is that both languages support SOM bindings.

The first set of language bindings is already available in the OS/2 2.0 toolkit for C. Additional language bindings under development include C++, Smalltalk, COBOL, and others.

IBM is actively encouraging various language vendors to directly support SOM bindings. IBM is also working with several different standardization bodies to make SOM as widely accepted as possible.

FIGURE 1.

SOM run-time class repository and language bindings



LISTING 1.

```
include <somobj.sc>

class:
  Student;

parent:
  SOMObject;

data:
  char id[16];    /* student id  */
  char name[32];  /* student name */

methods:

  void setUpStudent(char *id, char *name);
  --sets up a new student.

  void printStudentInfo();
  --prints the student information.

  char *getStudentType();
  --returns the student type.

  char *getStudentId();
  --returns the student id.
```

LISTING 2.

```
#define Student_Class_Source
#include ''student.ih''

static void setUpStudent(
   Student *somSelf, char *id, char *name)
{
   StudentData *somThis = StudentGetData(somSelf);
   strcpy(_id, id);
   strcpy(_name, name);
}
static void printStudentInfo(Student *somSelf)
{
   StudentData *somThis = StudentGetData(somSelf);
   printf('' Id       : %s \n'', _id);
   printf('' Name     : %s \n'', _name);
   printf('' Type     : %s \n'', _getStudentType(somSelf));
}
static char *getStudentType(Student *somSelf)
{
   StudentData *somThis = StudentGetData(somSelf);
   static char *type = ''student'';
   return (type);
}
static char *getStudentId(Student *somSelf)
{
   StudentData *somThis = StudentGetData(somSelf);
   return (_id);
}
```

The box in Figure 1, with C pointing to the run-time repository, represents a binding that allows classes to be defined in C and added to the SOM run-time repository. The box from the run-time repository to C is a binding that allows C to use classes from the repository regardless of the language in which the classes were originally written.

Because SOM is designed to work well with many languages and because it is focused on packaging issues, we think of SOM as packaging technology rather than language technology. The first major object-oriented library packaged with SOM, the Workplace Shell of OS/2 2.0, is already available.[4]

Defining classes in SOM

Creating class libraries in SOM using the C bindings is a three-step process. The class designer defines the class interface, then implements the class methods, and, finally, loads the resulting object code into a class library. Clients can use these classes directly, make modifications to suit their specific purposes, or add entirely new classes of their own.

In SOM, we define a class interface by creating a class definition file. We will give a basic example here and defer more detailed discussion of the many keywords and options to the SOM manuals.[5]

The class definition file is named with a .CSC extension. In its most basic form, the class definition file is divided into the six following sections: include, class name, parent information, data, and methods.[6]

Listing 1 shows a class definition file, student.csc, describing a *Student* class. In a future release of SOM, IBM intends to support the OMG CORBA IDL specification for defining classes as well as the object interface definition language (OIDL) shown here.[7]

Writing methods

Class methods are implemented in the class method implementation file. Each method defined in the method section of the class definition file must be implemented. The methods can be implemented in any language that offers SOM support, which for now is only C. The student class method implementation file, student.c, is shown in Listing 2.

The method code looks much like standard C, with a few differences. First, each method takes as its first parameter a pointer (*somSelf*) to the target object. This parameter is implicit in the class definition file but is made explicit in the method implementation.

Second, each method starts with a line setting an internal variable called *somThis*. This is used by macros within the SOM header file.

Third, names of data elements of the target object are preceded by an underscore character. The underscored name turns into a macro defined in the header file, part of the package SOM offers to shield method developers from the details of memory layout.

Fourth, methods are invoked using an underscore syntax. This underscored name turns into a macro invocation that shields programmers from having to understand the details of method resolution. Methods are always passed, as their first parameter, a pointer to the target object. This can be seen in the method *printStudentInfo()*, which invokes the method *getStudentType()* on its own target object.

The process of creating a class method implementation file can be greatly speeded up by the SOM compiler, which takes a .CSC file and generates a valid method implementation file lacking only the body of the methods. The body is then filled in by the class implementor. For the student example, the SOM compiler would create a file similar to that in Listing 3.

Deriving new SOM classes

Inheritance is an important mechanism for achieving code reuse in object-oriented libraries.[8]

In the following example, *GraduateStudent* is derived from *Student*, its base, or parent, class. A derived class automatically picks up all characteristics of the base class. A derived class can add new functionality through the definition and implementation of new methods. A derived class can also redefine methods of its base class, a process called overriding.

GraduateStudent adds *setUpGraduateStudent()* to those methods it inherits from *Student*. It overrides two other inherited methods, *printStudentInfo()* and *getStudentType()*. It inherits without changing *setUpStudent()* and *getStudentId()* from the *Student* base class. The class definition file for *GraduateStudent*, graduate.csc, is shown in Listing 4. The

LISTING 3.

```
#define Student_Class_Source
#include ''student.ih''

static void setUpStudent(
   Student *somSelf, char *id, char *name)
{
   StudentData *somThis = StudentGetData(somSelf);
}
static void printStudentInfo(Student *somSelf)
{
   StudentData *somThis = StudentGetData(somSelf);
}
/* ...and so on for the other methods. */
```

LISTING 4.

```
include <student.sc>

class:
  GraduateStudent;

parent:
  Student;

data:
  char  thesis[128];    /* thesis title */
  char  degree[16];     /* graduate degree type */

methods:
  override  printStudentInfo;
  override  getStudentType;
  void  setUpGraduateStudent(
      char *id, char *name, char *thesis, char *degree);
```

LISTING 5.

```
#define GraduateStudent_Class_Source
#include ''graduate.ih''

static void printStudentInfo(GraduateStudent *somSelf)
{
   GraduateStudentData *somThis = GraduateStudentGetData(somSelf);
   parent_printStudentInfo(somSelf);
   printf(''    Thesis     : %s \n'', _thesis);
   printf(''    Degree     : %s \n'', _degree);
}
static char *getStudentType(GraduateStudent *somSelf)
{
   static char *type = ''Graduate'';
   return (type);
}
static void setUpGraduateStudent(
   GraduateStudent *somSelf, char *id, char *name,
   char *thesis, char *degree)
{
   GraduateStudentData *somThis = GraduateStudentGetData(somSelf);
   _setUpStudent(somSelf,id,name);
   strcpy(_thesis, thesis);
   strcpy(_degree, degree);
}
```

LISTING 6.

```
#include ''student.h''
#include ''graduate.h''

main()
{
  Student *mark = StudentNew();
  GraduateStudent *jane = GraduateStudentNew();

  _setUpStudent(mark,''399542'',''Mark Smith'');
  _setUpGraduateStudent(jane, ''423538'', ''Jane Brown'',
              ''Code Optimization'',''Ph.D.'');
  _printStudentInfo(mark);
  _printStudentInfo(jane);
}
```

method implementation file for *GraduateStudent*, graduate.c, is shown in Listing 5.

Often, an overridden method will need to invoke the original method of its parent. For example, the *printStudentInfo()* for *GraduateStudent* first invokes the *Student* version of *printStudentInfo()* before printing out the *GraduateStudent*-specific information. The syntax for this is *parent_MethodName*, as can be seen in the *printStudentInfo()* method shown in Listing 5.

The SOM client

Now let's see how a client might make use of these two classes in a program. As we look at the program in Listing 6, we can discuss how objects are instantiated or created in SOM and how methods are invoked.

A class is instantiated with the method *classNameNew()*, which is automatically defined by SOM for each recognized class. Methods are invoked by clients just as they are inside SOM methods. The first parameter is the target object. The remaining parameters are whatever information is needed by the method. The only odd feature is the underscore preceding the method name that turns what looks like a regular function call into a macro defined in the .h file.

When run, this program gives this output:

```
Id       : 399542
Name     : Mark Smith
Type     : Student
Id       : 423538
Name     : Jane Brown
Type     : Graduate
Thesis   : Code Optimization
Degree   : Ph.D.
```

In the printout, we can see the override resolution at work in the different styles of output for *Students* and *GraduateStudents*. The *printStudentInfo()* method that a *Student* responds to is different than the method that a *GraduateStudent* responds to, as we can see from the preceding printout.

Class objects

Every class with at least one instantiated object has exactly one associated class object. Every instantiated object maintains a pointer to its class object, which it can retrieve through the method *somGetClass*. This method is inherited from *SOMObject*, a class from which every class is derived either directly or indirectly. In the program, *student1* and *student2* are both objects of class *Student*, and therefore both share the same class object. Similarly, *grad1* and *grad2* are both objects

of class *GraduateStudent*, and therefore both share another class object different than the class object shared by *student1* and *student2*. This relationship is shown pictorially in Figure 2.

In addition to accessing class objects via the *somGetClass* method, you can access a class object through the SOM-defined macro *_className*. For *Student*, this macro is *_Student*. This macro returns a pointer to the appropriate class object, if it exists, or zero, if it does not exist. SOM instantiates the class object the first time an object of its controlled class is instantiated. The *_Student* will, therefore, return zero if no *Student* objects have yet been instantiated.

The class object is quite a useful object. The class of a class object is either *SOMClass* or some class derived from *SOMClass*. A variety of methods are defined for *SOMClass*, and these are all documented in the class definition file for *SOMClass*, somcls.sc, and described in the SOM documentation. There are methods that show size of the class, parent of the class, whether the class is derived from some other class, and whether the class supports a specific method.

One of the *SOMClass* methods is *somGetName()*, which returns a pointer to a character string containing the name of the class for which this is the class object. For the *Student* class object, this string would be "Student". Another interesting *SOMClass* method is *somNew()*, which returns a newly instantiated object of the appropriate class. For the *Student* class object, the newly instantiated object would be of the *Student* class.

You can change the class of a class object through standard inheritance mechanisms. This is useful for adding the equivalent of C++ constructors and otherwise extending the capabilities of class objects.[9]

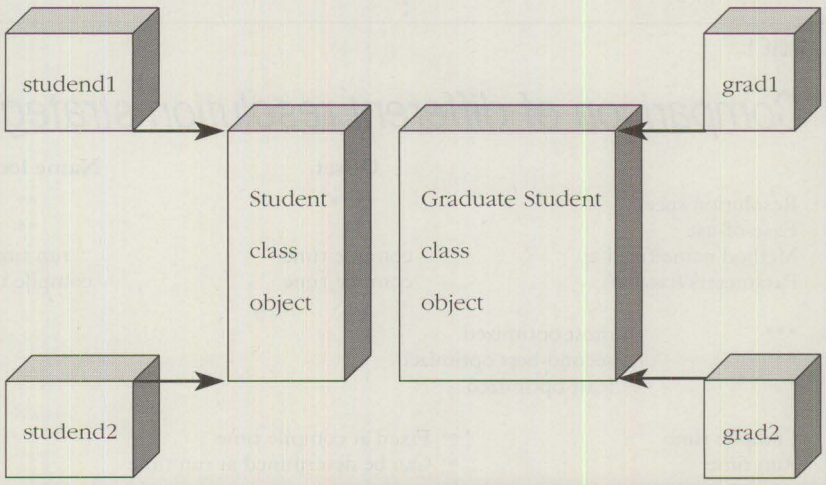
Method resolution

One of the features of SOM that allows bindings to be created for so many different languages is its concept of flexible method resolution.[10,11] This same flexibility can be exploited by the C programmer to choose the resolution most appropriate for a given task.

SOM supports three different method resolution strategies: offset, name lookup, and dispatch. If we compare flexibility, ease-of-use, and resolution speed, we find that each has a different optimization pattern. These differences are summarized in Table 1.

FIGURE 2.

Relationship between objects and class objects



This ability to choose from one of three resolution strategies is an important feature of SOM and allows SOM to be used in a wide variety of programming situations. At one end of the spectrum, SOM supports the highly efficient but inflexible offset resolution, similar to the virtual mechanism of C++.[12] At the other end, SOM supports the highly flexible dispatch resolution, allowing methods to be resolved when absolutely no compile time information is available.

Comparison to C++

In this section, we will compare some SOM features to those of the most widespread object-oriented programming language, C++.

SOM has many similarities to C++. Both support class definitions, inheritance, and overridden methods (called virtual methods in C++). Both support the notion of encapsulation. But where C++ is designed to support stand-alone efforts, SOM is primarily focused on the support of commercial quality class libraries. Most of the differences between SOM and C++ hinge on this issue.

C++ supports programming in only one language: C++. SOM is designed to support many languages. Rather than a language, SOM is really a system for defining, manipulating, and releasing class libraries. SOM is used to define classes and methods, but it is left up to the implementor to choose a language for implementing methods. Most programmers will, therefore, be able to use SOM quickly without having to learn a new language syntax.

C++ class libraries are version dependent. SOM class libraries are version independent. When a new C++ class library is released, client code has to be fully recompiled, even if the changes are unrelated to public interfaces.[13] SOM, unlike C++, directly supports the development of upwardly compatible class libraries.

C++ provides minimal support for implementation hiding or encapsulation. C++ class definitions, which must be released to clients, typically include declarations for the private data and methods. This information is, at best, unnecessarily detracting, and at worst, proprietary. In SOM, the client never has to see such implementation details. The client need see only the .sc files, which, by definition, contain only public information.

C++ has limited means of method resolution. SOM offers several alternatives. Like C++, SOM supports offset method resolution, meaning that methods are located via offsets in dispatch tables that are set up once and for all at compile time.

Unlike C++, SOM also offers facilities for resolving methods at run time. Name lookup resolution lets a client ask for a pointer to a method by method name. Dispatch resolution allows a client to package parameters at run time for dispatching to a method, a technique that allows SOM to be integrated into interpreted languages, such as Smalltalk.

SOM and C++ differ in their notion of class. In C++, the class declaration is very similar to a structure declaration. It is a compile-time package with little run-time significance. In SOM, the class of an object is an object in its own right. This object is an instantiation of another class, called the metaclass. The class object supports a host of useful methods that has no direct parallels in C++, such as *somGetName()*, *somGetParent()*, and *somFindMethod()*.

SOM should not be seen as a competitor to C++, but as an enhancement to C++. When C++ bindings are available, C++ programmers will be able to package their C++ classes to give them language independence and the other advantages of SOM packaged class libraries. ■

TABLE 1.

Comparison of different resolution strategies

	Offset	Name lookup	Dispatch
Resolution speed	***	**	*
Ease-of-use	***	**	*
Method name fixed at	compile time	run time	run time
Parameters fixed at	compile time	compile time	run time

*** = most optimized
** = second-best optimized
* = least optimized

Compile time = Fixed at compile time
Run time = Can be determined at run time

Acknowledgements

SOM is the work of many people. Mike Conner developed the initial idea and implementation and continues to lead the overall design of SOM. Andy Martin designed the SOM class interface language and designed and implemented the class interface compiler. Larry Raper implemented many features of the run-time library and ported SOM to OS/2. Larry Loucks provided close technical tracking and was instrumental in focusing the effort. Other SOM developers include Liane Acker, Nurcan Coskun, Scott Danforth, and Simon Nash. The project is managed by Jerry Ronga.

Roger Sessions has a B.A. from Bard College and an M.E.S. in database systems from the University of Pennsylvania. He has authored many articles and has lectured throughout the world on object-oriented programming, C++, and SOM. He is working on persistent SOM frameworks and previously worked with high-performance relational databases and object-oriented storage systems. He can be contacted at roger@vnet.ibm.com.

Nurcan Coskun has a B.S. in industrial engineering from Middle East Technical University and an M.S. and Ph.D. in computer science from the University of Missouri-Rolla. She is now working on object-oriented programming environments and previously worked on the OS/2 database manager. She can be contacted at nurcan@ausvm1.iinus1.ibm.com

References

1. Sessions, Roger. *Class Construction in C and C++: Object-Oriented Programming Fundamentals.* Englewood Cliffs, N.J.: Prentice-Hall, 1992.

2. Coskun, Nurcan, and Roger Sessions. "Object-Oriented Programming in OS/2 2.0," *IBM Personal Systems Developer*, Winter 1992.

3. Harkey, Dan, and Robert Orfali. *Client/Server Programming with OS/2 2.0*, 2nd Edition. New York, N.Y.: Van Nostrand Reinhold, 1992.

4. Wright, Mary. "The OS/2 Workplace Programming Interface," *IBM OS/2 Developer*, Summer 1992.

5. "OS/2 2.0 Technical Library System Object Model Guide and Reference," IBM Doc S10G-6309.

6. Coskun, Nurcan, and Roger Sessions. "Object-Oriented Programming in OS/2 2.0," *IBM Personal Systems Developer*, Winter 1992.

7. *The Common Object Request Broker: Architecture and Specification*, Object Management Group. CORBA, Framingham, Mass.

8. Sessions, Roger. *Class Construction in C and C++: Object-Oriented Programming Fundamentals.* Englewood Cliffs, N.J.: Prentice-Hall, 1992.

9. Coskun, Nurcan, and Roger Sessions. "Class Objects in SOM," *IBM OS/2 Developer*, Summer 1992.

10. Coskun, Nurcan, and Roger Sessions. "Method Resolution in SOM," *IBM OS/2 Developer*, Spring 1993.

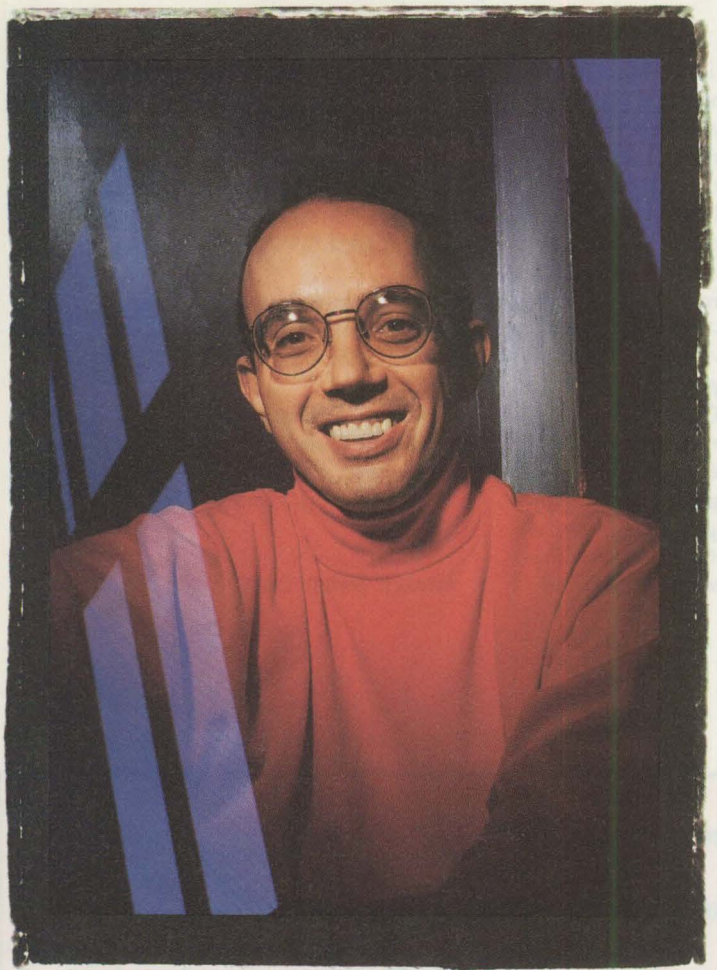
11. Coskun, Nurcan. "Using Name Lookup Resolution in SOM," *IBM OS/2 Developer*, Spring 1993.

12.Sessions, Roger. *Class Construction in C and C++: Object-Oriented Programming Fundamentals.* Englewood Cliffs, NJ: Prentice-Hall, 1992.

13. Ibid.

Artwork: Dwight Been

"Why I



Patrick Pearce, President, Chief Application Architect, Life Care Development Corp.

"Time is money."

The advantages of OS/2® are clear. "At Life Care Development Corp., we create applications for sale to physicians, psychiatrists and drug counselors for tracking patient and insurance information, and medicine and treatment goals. We make use of OS/2's inherent development capabilities like the REXX language as well as WorkFrame/2 (IBM's development environment), C Set/2 compiler and Borland ObjectVision®. For us, OS/2 has meant heightened productivity, shortened development time and improved quality of product."

Work in a customizable object-oriented environment without constraints. Enjoy true preemptive multitasking, unlike what you get with Windows™ and other DOS extenders. "With OS/2, I can reliably run several development applications at the same time: edit in one window, compile in another, link in a third and test in a fourth. I'm amazed how quickly I can compile a program while printing a copy of the source code." OS/2 gives you the capability to have multiple configurable sessions in which to build and test your applications.

ObjectVision

"OS/2 is easier to get into."

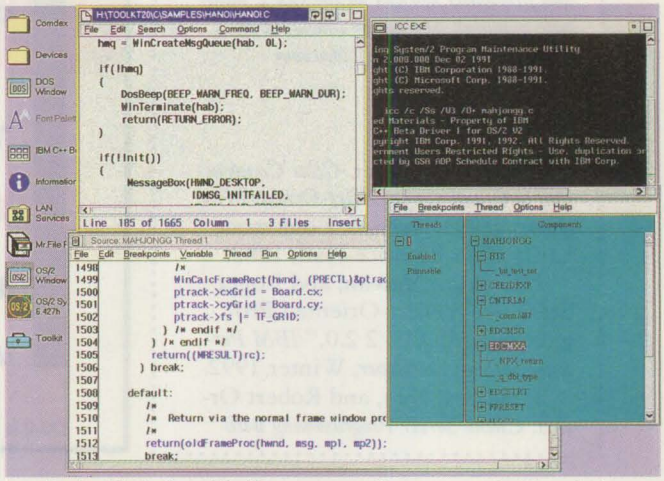
OS/2 Crash Protection™ helps you lose your fear of crashing and rebooting. If one app goes down due to a bug, the rest you're working on won't. OS/2 isolates the failure, letting you fix it and restart it without affecting other apps. Dynamic Link Libraries allow applications to share common functions, making them smaller and easier to maintain.

1992
– WINNER –
PC Magazine Award
for Technical Excellence

OPERATING SYSTEMS AND SOFTWARE STANDARDS
OS/2, Version 2.0
IBM Corporation

"I'll never go back."

"I may be a small ISV, but IBM has always treated me like a big fish." IBM's valuable technical service and marketing support includes OS/2 Support Line, IBMLink, the IBM OS/2 bulletin board system and several OS/2 developer forums on



In the Workplace Shell,™ you can edit source code files while compiling and debugging in the background.

don't do Windows anymore."

The no-comparison comparison chart.

	Windows 3.1	OS/2
Virtual memory limit	4 x physical	512MB (disk space)
Memory model	Segmented (64KB)	Flat memory objects
APIs	16 bit	Full 32 bit
Multitasking—DOS apps	Time slicing	Pre-emptive time slicing
Multitasking—Windows/PM apps	Cooperative	Pre-emptive
Priority	Static (set by user)	Dynamic
Dispatchability	Process	Thread
System services	Serial	Parallel
Protection between apps	Unprotected	Protected
Kernal protection—DOS/Win/PM apps	Unprotected	Protected
File system	FAT	Enhanced FAT and installable file systems (HPFS, CD-ROM)
User interface	Windowed	Object oriented

CompuServe®. "If I run into a problem, the OS/2 Developer Assistance Program is there to help."

The 32-bit operating system lets you break through the 64K code segment barrier and convert to a flat memory model with up to 512MB of memory per session for writing code. "Writing is easier and faster than ever—and bugs have never been easier to uncover and zap."

"I'm actually having fun again."

But the best reason for leaving Windows and other DOS extenders is the opportunity to develop truly revolutionary OS/2 applications. You could say OS/2 has closed the door on Windows. For the free white paper on why OS/2 is the developer's platform of choice, or for more information, call **1 407 982-6408.**

IBM®

Circle No. 805 on Inquiry Card

Container Yourself!

OS/2's container control is a uniquely flexible and powerful way to hold and display your objects.

by Randy Branham

A container is an OS/2 2.0 system-provided control that holds and displays other objects. On the screen, it looks no different than any other standard window. But this is where looks can be deceiving. It is an extremely flexible window, offering a wide variety of views to display and manipulate information in a relatively simple package. With a little bit of work, anyone can use this control while programming.

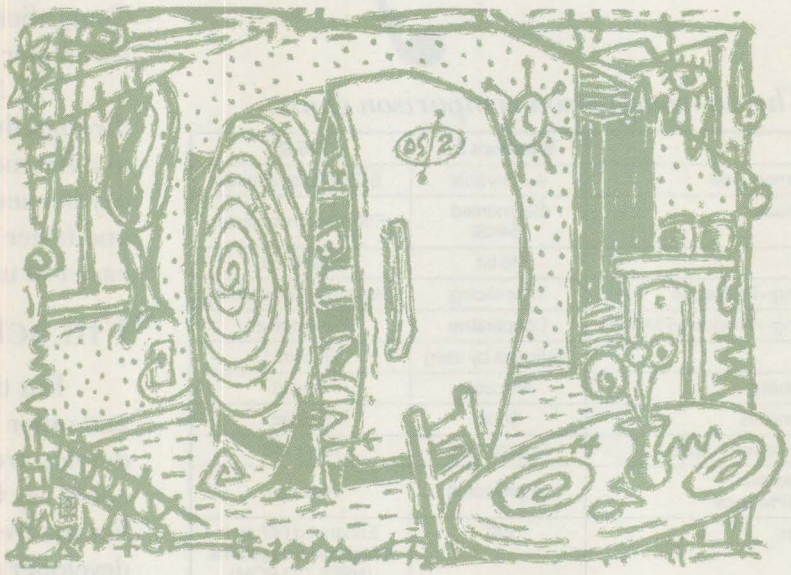
The creation of a container is rather easy. It's a standard window class that has some relatively simple attributes and styles. The styles control how objects within the container are selected, who positions the objects, whether the objects can be changed, and how we define the objects to the container. The attributes define how information is displayed within the container; whether it has titles, separators, and other lines; and who draws the container. Starting with this understanding, we're off to the world of programming containers.

Container styles

The styles are the first area of concern, and they effect the entire container. First, there are the selection styles: *CCS_EXTENDSEL*, *CCS_MULTIPLESEL*, and *CCS_SINGLESEL*. All these precipitate how objects within the container are selected, but all three have very different behavior. *CCS_SINGLESEL* allows only one object within the container to be selected at any time. The other two allow for differing types of multiple selection.

Second is the read-only style. *CCS_READONLY* prevents the user from opening edit windows anywhere within the container. If there is no user-modifiable data within the container, *CCS_READONLY* is very helpful. When we want the container to automatically position objects within it, we use *CCS_AUTOPOSITION*. This makes life a little easier when we're displaying simple text and icons.

The attributes define a little more about the container, titles, title positioning, and the appearance



LISTING 1.

```
hcont = WinCreateWindow( hwndparent
                  , WC_CONTAINER
                  , &CONTAINER_TITLE[0]
                  , CCS_AUTOPOSITION  CCS_EXTENDSEL
                  , 100, 100
                  , 100, 100
                  , hwndowner
                  , HWND_TOP
                  , id
                  , 0
                  , 0 );
```

LISTING 2.

```
WinSendMsg ( hcont
           , CM_SETCNRINFO
           , &cnrinfo
           , MPFROMLONG(CMA-CNRTITLE  CMA-FLWINDOWATTR));
```

of lines in one of the views. With *CA_CONTAINERTITLE* we can put a title within the container. This title appears on the top line and can be positioned to the left, right, or center, depending on *CA_TITLELEFT*, *CA_TITLERIGHT*, or *CA_TITLECENTER*. *CA_TITLEREADONLY* makes the title read-only. It is just like the control style CCS_READONLY, but it makes only the title read-only. If this attribute is not specified, the titles within the container can be edited and changed by the user, just like any other field, which can be particularly disastrous if it isn't planned for in advance.

Creating the container

Once the styles and attributes are chosen, the actual creation of the container is relatively easy. By choosing the *WC_CONTAINER* class, we can create containers by making a standard window call, as shown in Listing 1.

Here, we are creating a container with the control styles of *CCS_AUTOPOSITION* and *CCS_EXTENDSEL*. To specify the attributes we desire for the container, we need to populate the *flWindowAttr* field of a *CNRINFO* structure and pass this structure to the container on a *CM_SETCNRINFO* message. This can be seen in the example immediately following the creation of the container. It looks similar to Listing 2.

In this example, we are changing the attributes on the container and putting a title within the container. A pointer to the populated *CNRINFO* structure is in the first message parameter. This is where we have populated two fields, *flWindowAttr* and *pszCnrTitle*, with the attributes we've selected and a pointer to the title we wish to display.

The second message parameter is where we tell the container what fields within the *CNRINFO* structure we've updated. As a programming habit, I would recommend querying the container for the current *CNRINFO* by sending it a *CM_QUERYCNRINFO* message before replacing it. With that, we've created a basic container that has a title.

Now that we have a basic container, let's put some records in it. Before we can do this, we need to define a record structure that we can use for examples. My example is shown as the structure in Listing 3. This structure is pretty simple with one unique structure within it, *RECORDCORE*. It links our world and the world of the container. It is very important that this be the first area within our sample record structure.

By making it first, a pointer to our record doubles as a pointer to a record core structure that the container desires and a pointer to a *SAMPLERECORD* that we desire. This is not the only type of *RECORDCORE*. Actually, two types of *RECORDCORE* structures exist. The other is a *MINIRECORDCORE*, a simpler and smaller version of *RECORDCORE* that allows for only one icon and

LISTING 3.

```
typedef struct _samplerecord {
                        RECORDCORE  recordcore;
                        HPOINTER    hptrIcon;
                        PSZ         pszString;
                        ULONG       ulNumber;
                        CDATE       cdate;
                        CTIME       ctime;
                        } SAMPLERECORD, *PSAMPLERECORD;
```

LISTING 4.

```
/* ask the container to allocate a region for us */
size = (LONG) (sizeof(SAMPLERECORD) - sizeof(RECORDCORE));
psamplerecord = WinSendMsg ( hcont
                , CM_ALLOCRECORD
                , (MPARAM)size
                , (MPARAM)1 );
```

one piece of text associated with all the basic container views.

To use *MINIRECORDCORE* instead of *RECORDCORE*, you just specify the *CCS_MINIRECORDCORE* style bit when the window is created. The other fields in the structure define all the other data we wish to display.

When adding a record, the first thing we need to do is ask the container to allocate space for it. Listing 4 shows how to send the container a *CM_ALLOCRECORD* message with the additional size we need allocated for our information.

The example does this for one record, but several records could have been asked for if needed. We can then populate the returned area with the appropriate data, which includes populating the appropriate fields within the *RECORDCORE* structure. The example loads an icon, populates the string, date, and time fields, and then initializes some of the fields in the *RECORDCORE*.

Before we can add the record, we must tell the container where and how to add it. This is done with a *RECORDINSERT* structure. This area will tell the container where you wish the desired record to be placed (in the front or back) and how many records you are inserting.

We can now send a *CM_INSERTRECORD* to the container, passing it the pointers to these two structures. You can see all of this in the *ADDRECORD* function within the example. *ADDRECORD* allocates the space for one additional record, initializes both structures, and inserts the record with the call shown in Listing 5.

Here, *hcont* is the handle to the container, *CM_INSERTRECORD* is the message, *psamplerecord* is a pointer to the sample record we wish to insert, and *rci* is the *RECORDINSERT* structure defining how we wish this record inserted.

Deleting records

Deleting records is even simpler than adding them. We send a *CM_REMOVERECORD* to the container with three pieces of information. First is a pointer to the array of *RECORDCORE* structures to be removed. Second is the number of pointers. If zero is passed, all the records in the container will be deleted. Third are the flags telling the container how to remove the records. *CMA_FREE* tells the container to automatically free the storage used by *RECORDCORE*. *CMA_INVALIDATE* is the other flag. It tells the container what should be done with the remaining records in the container, whether they are invalidated, and if any repositioning should occur. The example deletes all selected records from the container, freeing them along the way.

Deleting records introduced us to a new area: querying the container and the current emphasis of its records. A record within a container can have three types of emphasis: cursored, which has a little dotted box around it; in-use, which has diagonal hashing through it; and selected, which is highlighted. The container takes care of the cursored and highlighted attributes for us whenever we click within it and automatically applies the selection style to update the record's current state. In-use messages must be sent by the application program by sending *CM_SETRECORDEMPHASIS* for the record in question to the handle of the container. This is usually done whenever a record is double clicked to give visual feedback that the double click has been received and is being processed.

At this point, we have a fully

functional container to which we know how to add, delete, and emphasize records, but we haven't added the user interface to know when we wish to do this. A pop-up menu will be our answer. When the user requests a menu with a single click of the minor mouse button (the right mouse button for right-handed people), the container receives a *WM_CONTEXTMENU* request, which it passes on to its owner in the form of a *WM_CONTROL* message *CN_CONTEXTMENU*.

With this message, you also get the ID of the container sending this message and a pointer to a *RECORDCORE* structure over which the mouse operation occurred. If this action occurred over empty space (usually called white space), this pointer will contain a null value.

The pointer is the key item. With it, we can interrogate the record and pop up a different menu for each record in the container if we wish. The actual menu is displayed with a *WinPopupMenu* call. To use this function, we must have first loaded or created a handle to a menu using *WinLoadMenu* or *WinCreateMenu*. This is demonstrated in the sample program.

It receives the *CN_CONTEXTMENU*, saves the pointer that the request is made for, and pops up the menu. The menu handle that is used for this request was first loaded in the *WM_CREATE* process with a *WinLoadMenu* call. Because the example has a homogeneous set of sample records, it only loads and pops up one type of menu.

By taking a look at the menu contained in the sample, we see that it does more than just add and delete records. It also allows us to change the view of the container. By default, the container gives us five views: icon, name, text, detail, and tree. Icon and name view are basically the same, just arranged a little differently. They both provide for an icon and some text to be displayed either underneath or beside the icon. Text view is even simpler. It displays only the text associated with record.

Other views

Detail and tree view are a little more complex. With tree view, you must define the parent of the record you are inserting by supplying a pointer to its *RECORDCORE* and a *TREEITEMDESC* structure, which contains handles to the bitmaps or icons that are to be used in the expanded and collapsed states. This is demonstrated by the OS/2 drives object. Detail view is probably the most complex. With it, you must define the number of columns to be displayed and the actual column information.

All these views need not display the same text. The text displayed in the name view need not be the same text displayed within the tree or text views. This allows for quite a bit of customizing.

Another item in the menu of the example is a choice to *Arrange*. If we were to remove the style bit *CCS_AUTOARRANGE*, our objects would always be added to the same coordinates as specified on the *CM_INSERTRECORD* message, and our deletes would leave holes where removed records were displayed. Even if we resized the window, the objects would stay in their current positions. Selecting this option (*Arrange*) would cause the icon view of the container to arrange itself, filling the container from left to right, top to bottom.

If the icons ever get out of order, we could realign them with this function. With *CCS_AUTOARRANGE*, this choice should never be necessary, as the icons would arrange themselves automatically, but it has been included just to make sure.

The example creates all of its detail view in one function. It starts off by asking the container to allocate space for the columns to be displayed, as shown in Listing 6.

In this case, the container has five columns. Returned is a pointer to a linked list of *FIELDINFO* structures that we use to delineate the columns. In defining these columns, we must remember that there are five basic types: bitmaps or icons, strings, numbers, dates, and times. The type of column depends on which attribute is specified in the *flData* field of the *FIELDINFO* structure.

We can also specify whether or not this field is separated from others, whether it's read-only, how its text should be displayed (horizon-

LISTING 5.

```
WinSendMsg ( hcont
           , CM_INSERTRECORD
           , (MPARAM)psamplerecord
           , (MPARAM)&rci );
```

LISTING 6.

```
pfi = WinSendMsg ( hcont
          , CM_ALLOCDETAILFIELDINFO
          , (MPARAM)NUM_COLS
          , NULL );
```

tally and vertically), and whether or not the column is visible. If the column is set to handle strings, the user may directly edit this field by pressing the Alt key at the same time as the major mouse button (left button for right-handed people).

This editing capability can be turned off by simply specifying *CFA_FIREADONLY* in the *flData* field of the *FIELDINFO* structure. The next field, *flTitle*, is where the attributes for any titles to be displayed are specified. Standard right, left, and center for horizontal and vertical positioning are available, but the title does not have to be text. If the attribute *CFA_BITMAPORICON* is set, the column heading can be a picture instead of words.

Another field in the *FLDINFO* structure is the *cxWidth*. This is where a fixed width is given to a column. If no value is specified, the container will automatically size the column to the length of the largest field in the column. After all the *FIELDINFO* structures have been initialized (the date and time columns use the *CDATE* and *CTIME* layout), we use another structure to tell the container where these columns are to be inserted. This would be the *FIELDINFOINSERT* structure. It tells the container where to insert the columns and whether or not to redraw the container after inserting them. Now, it's just a matter of sending the container a *CM_INSERT* message with the proper pointers (one to the first *FIELDINFO* and another to the *FIELDINFOINSERT*), and it will return the number of *FIELDINFO* structures in the container.

The example adds one more item to the container, a vertical split-bar. A split-bar divides the container into two windows that can be scrolled independently whenever the container is in detail view. It can also be moved horizontally to resize both windows. This is very helpful when some type of reference information is needed while scrolling through a long list of columns. To add a vertical split bar to a container, you must specify the horizontal distance relative to the origin of the container to position the split-bar and the last column in the left half of the detail view. The vertical split-bar is added after the first column with the *CM_SETCNRINFO* message.

Another point of interest is that the example or what's shown in the example allows direct editing of the string field. To begin the direct editing, the user holds the Alt key down while pressing the major button on the mouse over the string field of a record. When this occurs, the container sends the program *CN_BEGINEDIT*, *CN_REALLOCPSZ*, and *CN_ENDEDIT* messages.

The *CN_BEGINEDIT* informs the program that direct editing of a field is starting. Once a direct editing session has started, no messages should be sent to the container until after the *CN_ENDEDIT* message has been received. Unpredictable results may occur if we do otherwise. When the editing session is about to close, we receive *CN_REALLOCPSZ* and can test to see if there is sufficient memory for the container to copy the new string over the old string. The example then *frees* the old string and *mallocs* a new area to accept the new string. If we were to return *FALSE* instead of *TRUE*, the new string would simply be discarded by the container. *CN_ENDEDIT* means that the edit session has finished, and we should update our container contents with the new value. We can once again send messages to the container.

Highlights

As you can see, the container is one of the most versatile controls offered. Myriad displays can be achieved through a relatively simple set of interfaces while, at the same time, limiting the dependence of the visual display on the physical data. It also opens a new area of interface design, giving users the capability to directly control their environment. I have briefly covered a few container highlights here, but this is not where it ends. Other areas such as drag and drop, sorting, filtering, and displaying mini-icons are definitely worth exploring. ■

Randy Branham is a consultant who has developed three-tiered development environments for five years. For the last 18 months , he has devoted his efforts entirely to OS/2 2.0. He can be reached on CompuServe at 71044,137.

Artwork: Donald Button

Unleash Your 486!

NDP Fortran is now six years old, yet people still call with 486 performance questions related to their use of 16-bit compilers. Therefore, we decided this was a good time to remake the case for 32-bit tools and languages.

To get full access to a 386 or 486 requires that it be placed in 32-bit protected mode. This is done by 32-bit operating systems and some Extenders. Running in 32-bit protected mode increases the 486's segment size from 64 Kbytes to 4 Gigabytes and breaks the 640K DOS barrier. This makes it possible to access arrays larger than 64K using the 486's 32-bit "flat" model, which runs 2 to 4 times faster than the huge model employed by 16-bit compilers. 32-bit tools also simplify ports. Programs written for mainframes often assume that integers and pointers are 32-bits long or depend on binary representations, especially when written in Fortran 66 and C. Our 32-bit NDP compilers eliminate porting headaches by providing the extensions you need to recompile your VMS, VS, Cray and MS codes. We also sell the IMSL, NAG and VAST II tools that frequently get bundled with mainframes.

Before you give up on your 486, give us a ring. Whether you are looking for the 5 megaflops your 486 produces by itself, the 12 megaflops it delivers with a Weitek 4167 or possibly the 80 megaflops we can coax out of our Number Smasher-860, you owe it to yourself to contact our excellent Tech Support Staff today. Microway has had the answers to your PC performance problems since 1982!

Phone: **(508) 746-7341**
Fax: **(508) 746-4678**

386, 486 and i860 Compilers

Microway's ***NDP*** family of 32-bit compilers generate globally optimized mainframe quality code that runs on the 386, 486 and i860. They require the use of 32-bit operating systems such as OS/2, UNIX, Xenix, Solaris, Coherent, DESQview/X and DPMI and VCPI DOS Extenders.

NDP Fortran[tm] is a full F77 with F66, DOD, VMS and MS extensions.

NDP CIC[tm]**++** includes a full C compiler that runs in both K&R and ANSI modes plus a C++ compiler that is Release 2.1 compliant.

NDP Pascal[tm] is a full ISO Level 1 Pascal with BSD extensions that can interface the NDP C runtime libraries.

NDP Language Pricing

Coherent versions use the Coherent tools and support the x87 family $295

MS-DOS versions include a VCPI virtual memory DOS Extender, a DPMI interface layer, support for the x87 and Weitek coprocessors, NDPLink, NDPLib and GREX - our DOS graphics library. The 486 version adds 486 code generation, royalty free DPMI and VCPI plus ClearView, Microway's symbolic debugger.

386 Release.....$595
486 Release $995

OS/2 releases include x87 support, a WorkFrame interface and our 486 extensions. These tools work with the IBM Linker. CIC++ includes the portions of the IBM Tool Kit needed to interface the PM API's $695

UNIX/Xenix 386/486 use the native tools on the system in question. V.3 and V.4 versions are available $1195

DESQview/X $1195

Coprocessors

Weitek	4167 25/33 $350/$795	3167 $200/250	
Cyrix	D8733/40..$99/129	S87..$79	EMC $250
Intel	387SX..$80	387DX..$85	287..$80
	OverDrive 25 MHz $475..	33 MHz $675	

i860 Supercomputers

QuadPuter-860 The world's most cost effective SuperComputer, the QuadPuter comes with four modules each of which has an i860 and two megabytes of local memory. The modules sit on an EISA base board that can be purchased with 8 or 32 megabytes of shared memory. Five QuadPuters have an aggregate throughput of a gigaflop! The card comes with one NDP language plus MAX-860.
From.....$9995

Number Smasher-860 our ISA card can be purchased with 8/32 megabytes of memory running at 33 or 40 MHz. Options include EISA and ISA Fifos, a P version with built-in Fifos for distributed parallel processing and a 24-bit graphics buffer. The card includes one NDP Language plus OS860. From.....$3995

NDP Fortran-860 along with our CIC++ and Pascal utilize advanced scalar code generation techniques designed to optimize the i860's RISC performance. The languages run on DOS, UNIX, OS/2 and i860 UNIX V.4 Workstations.....$1995

PPS-860 postprocessing scheduler, takes assembler output and converts scalar operations into pipelined. It also reschedules using dual instructions. Scalar speed ups in the range of 10 to 40% are produced $595

VAST-II from PSR is the same vectorizer that is used with Cray's.....$1495

AT Accelerators

FASTCache-X86/Plus[TM] — The *"Windows Solution"* - easily converts your 286 based PC into a 386/486 powered workaholic. The board takes up a slot in AT bus designs and has sockets for up to 16 megabytes of DRAM. It plugs into your 286 using a cable. The PS/2 version has no memory. All boards have a 16K four way cache and run at 25 MHz. Two processors are available, including the Cyrix CX486SLC.
From.....$399

486 Workstations

A Microway Black Tower is the ideal solution to your 486 needs. They feature industrial grade American power supplies, heavy duty cooling and easy access. All motherboards are carefully burned in before shipping and equipped with heavy duty power connectors. Our BX Towers are ideal for use as desktop workstations, file servers, RAID servers and CAD/CAM stations. They were originally engineered to house i860 arrays, which frequently get attached to SUN Networks as computational servers. We customize each system with the OS of your choice, including UNIX, OS/2, DOS and Windows. The base systems listed include case, 4 MB, tactile response keyboard, supply, floppy and mouse. The B^2T is our tall tower, the B^3 is a larger box that features front and side hinged panels. All but the 386 system use industrial supplies and the B^3 is available with a 12 slot EISA channel. To see why Microway systems have earned an international reputation for outstanding design, price/performance, value and reliability, ask for our **BX Series** Catalogue.

386B^{2}T-40 ISA 64K Cache	200W	$1395
486B^{2}T-33 ISA 64K "	250W	$2495
486B^{2}T-33 EISA 256K "	250W	$3395
486B^{2}T-50 EISA 256K "	250W	$3995
486B^3-50 EISA 256K "	350W	$4495
486B^3-66 EISA 512K & 12 Slots	350W	$5495

386, 486, i860 Libraries

NDP IMSL Microway compiled and validated version of the IMSL mainframe libraries, available for the 386/486 and i860.....$2,000

NDP NAG Microway compiled and validated NAG Workstation library, available for the 386/486 and i860. 386/486.....$995, i860.....$1495

KUCK & ASSOCIATES hand coded i860 libraries. DSP library does 1024 real FFT in 500 microseconds! **DSP**..$750 **BLAS**..$500

LAPACK and BLAS vectorized, sources included. 386/486.....$295 i860.....$495

Microway® *Technology you can count on!*

Research Park, Box 79, Kingston, MA 02364 USA (508) 746-7341 FAX (508) 746-4678
Kingston-Upon-Thames, U.K., 081-541-5466 Spain/Portugal 351-1-604-049
Germany 069-752023 Greece 30 12915672 Italy 02 7490749 Japan 047 423 1322 Poland 22-414115

Circle No. 806 on Inquiry Card

Notebook Bene

The notebook control is an innovative way to group collections of windows. Here's a step-by-step guide to programming it.

by Paul Montgomery

The notebook control was included in OS/2 2.0 to provide a mechanism for creating logical groupings of information and actions. Visually, the notebook resembles a spiral or hardbound book with tabs placed along the edges as indexes. Users can click on one of these tabs and go directly to the position in the book it represents.

To programmers, the notebook is a control that manages the Z-ordering of a set of windows. This control can be created and positioned just like any other control under OS/2. Its intent is generic; its use by programmers provides its usefulness and meaning to users.

I'm assuming that readers are familiar with Presentation Manager programming in general and have available the IBM C/Set 2 compiler and IBM OS/2 2.0 Toolkit.

What's a page?

A page in the notebook is actually a window defined by the programmer. The notebook controls the visibility of each window based on user selection of that page. The window is sized and positioned by the notebook control to take up the entire visible page whenever a user selects the page it represents.

The window can be anything from a simple control to a very complex window containing many other windows. The content of these windows is what gives meaning to the notebook. Take the OS/2 Desktop's settings notebook. This notebook is a collection of configuration options for the OS/2 desktop. Each page represents a different aspect to configure.

Some pages have groups of things, others have a single item. For instance, the View page is used to configure how the icons are displayed on the desktop. It has six radio buttons grouped into two group boxes, a font change group box, and a series of three buttons across the bottom.

All of these are placed on a frame window. This page could have been a dialogue brought up in response to a selection on a menu, but the designers concluded that

LISTING 1.

```
static HWND CreateNotebook(HWND parent,HWND owner,ULONG id,ULONG style)
   {
   HWND hbook;

   // Create the notebook and return its handle to the caller.
   hbook = WinCreateWindow( parent
            , WC_NOTEBOOK
            , ""
            , style
            , 0, 0
            , 0, 0
            , owner
            , HWND_TOP
            , id
            , 0
            , 0
            );

return hbook;
}
```

LISTING 2.

```
pwd->hbook = CreateNotebook ( hwnd
             , hwnd
             , NOTEBOOK_ID
             , BKS_SPIRALBIND
             | BKS_BACKPAGESBR
             | BKS_MAJORTABRIGHT
             | BKS_ROUNDEDTABS
             | BKS_STATUSTEXTLEFT
             | BKS_TABTEXTLEFT );
```

this page represented only one set of selections a user would have to make to configure the desktop. So they decided to group all the dialogues together into a notebook and let each page collect different categories of information.

Using a programmer-defined window for each page is a very powerful concept. By doing so, a page is only limited by the items that can be contained within windows, and, in Presentation Manager, that is anything.

For example, a notebook could be a collection of applications with each page being a different application. Or the notebook could be a dictionary with a tab for each letter of the alphabet. You could even use the notebook as a channel changer for a set of video windows. Each tab represents a different channel or selection of video. The notebook could be a portfolio of clip art with each page being one or more drawings. You could have the major tabbed pages be a description of the clip art to follow and the minor tabbed pages be the individual pieces of clip art. To do this, you would need three simple types of windows: a text window (really a window with an MLE on it), an art window (just a window with the *WM_PAINT* processing, putting a bitmap in the window), and a main client window that contains the notebook, creates the pages, and tells them what to display. The cover of the notebook (that is, the first page of the notebook) would also be a text window with an explanation of the contents of the entire notebook.

Creation of the notebook

The notebook control is a window like any other control under OS/2 (for instance, a button, a container, or a list box). It is created with a call to *WinCreateWindow* with the *pszClassName* parameter set to *WC_NOTEBOOK*. The notebook expects the *flStyle* parameter of the *WinCreateWindow* call to be filled with a set of flags that determine the visual aspects of the notebook. For instance, do you want a spiral or solid binding? Is the binding on the left, right, top, or bottom? Where are the tabs? They need to be on a different side from the binding. Are the tabs square? Rounded? Polygons? For this example, I will create a notebook with a spiral binding on the left and the major tabs on the right. The tabs will have rounded corners and the text in the tabs will be left-justified.

I have created a set of functions used to interact with the notebook. The first of these functions is *CreateNotebook*. It is shown in Listing 1.

The size parameters for the notebook are zero at this point. During the processing of the *WM_SIZE* message, the notebook will be sized to the current size of the client area of the window. This function is called with the parameters shown in Listing 2 to create this notebook.

The value returned in *pwd->hbook* is the *HWND* for the notebook. This call is done in the *WM_CREATE* processing of our main client window. We also want to set up a default size for the tabs in the notebook. This is done through the *BKM_SETDIMENSIONS* message.

We will set up a default size for the tab of 10 characters wide and a single character high. The tab size is based on the font metrics of the current font for the client window. This is obtained through a GPI call *GpiQueryFontMetrics*.

The *FONTMETRICS* structure obtained through this call has two fields that we need. One (*lMaxBaselineExtent*) we use for the height

LISTING 3.

```
BOOL SetTabDimensions(HWND hnb,USHORT width,USHORT height,USHORT type)
{
   return (BOOL) WinSendMsg ( hnb
                  , BKM_SETDIMENSIONS
                  , MPFROM2SHORT ( width, height )
                  , MPFROMSHORT ( type )
                  );
}
```

LISTING 4.

```
ULONG MakeNotebookPage(HWND hnb,ULONG pageid,USHORT style,USHORT order)
{
 return (ULONG) WinSendMsg ( hnb
         , BKM_INSERTPAGE
         , (MPARAM) pageid
         , MPFROM2SHORT( style, order )
         );
}
```

of the tab. Another *(lAveCharWidth)* we multiply by 10 to get the size of a 10-character-long tab. We then send the *BKM_SETDIMENSIONS* message to the notebook with a value of *lMaxBaselineExtent* for *usHeight* and a value of *(10 * lAveCharWidth)* for *usWidth*. *usType* is set to *BKA_MAJORTAB* for this call. We can then repeat the call for the minor tabs by just changing the *usType* message to *BKA_MINORTAB*.

In Listing 3, I have created a function, *SetTabDimensions*, to send the message to the *BKM_SETDIMENSIONS* message and on to the notebook. The width, height, and type parameters are set as I described previously. Once we have created the notebook, we are now able to insert pages.

Inserting a page

Inserting a page into a notebook is a multistep process. First, you must tell the notebook that you want to insert a page and where you want it inserted. You then associate a window with that page. After the page is inserted, you can give the tab a value or associate user data with the page. If you discover that the tab size you set up is not large enough to hold the text or bitmap you want, you may also resize the tabs.

The message used to insert a page is *BKM_INSERTPAGE*. This message has two parameters in which three pieces of information are given. The first parameter contains the ID of the page to insert. This ID must either be the ID of a previously inserted page or *NULL*. The second parameter is a combination of two *USHORT*s. The first is called the page style. Style in this context means: do you want the page automatically sized (*BKA_AUTOPAGESIZE*), do you want a major or minor tab on this page (*BKA_MAJOR* or *BKA_MINOR*), and do you want the status line on the bottom of the page to be displayed (*BKA_STATUSTEXTON*).

The second *USHORT* is the page order. This allows the programmer to specify where, in relation to the page ID given in the first parameter, to place the inserted page. The values available for use here are *BKA_FIRST*, *BKA_LAST*, *BKA_PREV*, and *BKA_NEXT*. If *BKA_FIRST* or *BKA_LAST* are used, the page ID in the first parameter is ignored. If *BKA_NEXT* or *BKA_PREV* are used, the page ID must be a valid ID, and the new page is inserted relative to the page specified.

The two *USHORT*s are combined into one parameter through the use of the *MPFROM2SHORT* macro. The return from the *BKM_INSERTPAGE* message is the page ID of the inserted page. Once again, I have created a function, shown in Listing 4, to do the mechanics of the *WinSendMsg*. The function is called with the parameters in Listing 5 to insert the first page into the notebook.

In the example, the first page created is our title page. At this point, the page has already been inserted, so now we need to create the window that is associated with this page, a text window. The text window is registered as the window class *TEXTCLASS* in the main function in the example. The *TEXTCLASS* window just creates an MLE and fills it from a text resource.

I have set up a relationship between the ID of the window and the ID of the resource to retrieve from the resource file. The ID of the window is the ID of the tab text for the page, and ID +1 is the ID of the text resource to fill the MLE with. ID +3 is the ID of a string that contains the number of art windows following this page.

First, we create an instance of this window and give it the ID of the text resource that we want it to display in its MLE. This is done through the *WinCreateWindow* call shown in Listing 6.

Once the window is inserted, the *BKM_SETPAGEWINDOW* command is used to associate a programmer-created window with the page. The message takes two parameters. The first is the page ID returned from the insertion of the page, and the second is the handle of the window. This is done with the function *SetNotebookPageHwnd*

shown in Listing 7, which is called with the parameters in Listing 8.

Now that we have a page with a tab, we need to tell the notebook what text to put on the tab. The tab text is added with the *BKM_SETTABTEXT* message. This message also takes two parameters. The first is the ID of the page to change the tab text on. The second is the text string to use. If the text you use for the tab is too long to fit in the space provided, it is truncated.

Where it is truncated depends on how you have configured the alignment of the text (this was done in the style flags during the creation of the notebook) and the size you have created for the tabs. For example, if you have used *BKA_TABTEXTCENTER* for the alignment, both ends of the text may be truncated. So at this point, we may need to send a *BKM_SETDIMENSIONS* message again to enlarge the tabs to fit the new text.

This process continues for each of the text pages and art pages.

LISTING 5.

```
// insert a page into the notebook
pageid = MakeNotebookPage ( hbook
   , NULLHANDLE
   , BKA_AUTOPAGESIZE | BKA_MAJOR
   , BKA_FIRST
   );
```

LISTING 6.

```
hTitlePage = WinCreateWindow( parent
           , "TEXTPAGE"
           , ""
           , WS_VISIBLE
           , 0, 0
           , 0, 0
           , owner
           , HWND_TOP
           , id
           , NULL
           , 0 );
```

LISTING 7.

```
BOOL SetNotebookPageHwnd ( HWND hnb, ULONG pageid, HWND hpage )
   {
   return (BOOL) WinSendMsg ( hnb
            , BKM_SETPAGEWINDOWHWND
            , (MPARAM) pageid
            , (MPARAM) hpage
            );
   }
```

When an art page is being inserted, we change the *BKA_MAJOR* in the page style to a *BKA_MINOR*, and we create a window of class *ARTCLASS*.

Interaction with the notebook

Once we have all the pages inserted into the notebook, the interaction with it becomes very simple. When users press a tab on a notebook, it sends a *WM_CONTROL* message with a notify code of *BKN_PAGESELECTED* to its owner (our client window). The *notifyinfo* field of the message points to a *PAGESELECTED* structure. This structure contains the handle of the notebook sending the message, the ID of the page being left, and the ID of the page being brought to the front.

The owner window can send any messages to the windows associated with these pages to notify them of the change. Owners can use this message to keep track of what page is on top, determine when to go get more data, or perform any other action that comes to mind. In our example, we do not need to track which window is on top so the program does nothing with this message. In fact, this application has nothing to do until the user closes the notebook. What could be simpler?

When a notebook is destroyed, any pages that are still in the notebook are destroyed, too. No notification is given that this is happening. It is up to the owners to remove the pages if they don't wish the window associated with the pages to be destroyed. It is also up to owners to delete any resources they may have associated with a page. Each page should also be able to handle the *WM_DESTROY* message and free any resources it may have allocated. Failure to do this will cause resources to stay around until the process is closed.

LISTING 8.

```
SetNotebookPageHwnd ( hbook
                    , pageid
                    , htitlepage );
```

Programmers can find complete definitions of the messages that a notebook can send and receive in the IBM Programmers Toolkit Presentation Manager Programming Reference on-line reference or in the *IBM OS/2 Technical Library* in printed form. ■

Acknowledgements

I would like to thank IBM for the work it has done to make OS/2 2.0 the outstanding product that it is. The notebook control is but one example of the enormous amount of effort and thought the programmers and designers at IBM have put into OS/2. A lot of people at IBM have put heart and soul into this product, and I think they deserve every accolade that can be accorded to them.

Paul Montgomery is president of TASP Group Inc., which specializes in OS/2 consulting and programming. He can be reached on CompuServe at 71500,3525.

Artwork: Dwight Been

Worldwide Developer Assistance Program

It even kills bugs.

It's fast relief from those annoying technical nits that can drive you buggy. It's answers to questions. Solutions to problems. It's help when you need it.

CompuServe

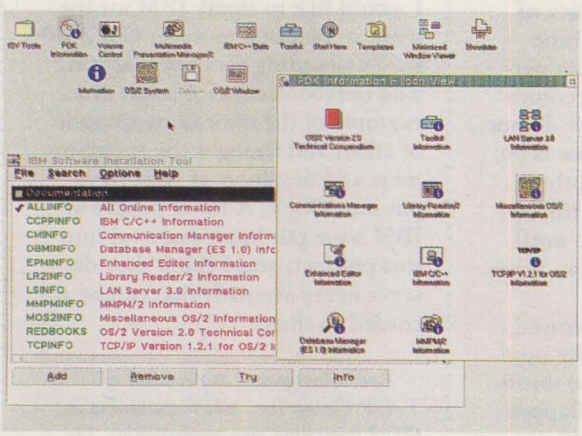
From development through generating market demand for your OS/2® software products, the Worldwide Developer Assistance Program (DAP) is there for you.

It's technical support and a database of common OS/2 questions and answers available through CompuServe® and other worldwide online systems. It's *IBM OS/2 Developer*, a magazine featuring technical and "how-to" articles on exploiting the rich features of OS/2. It's discounts on IBM PS/2® hardware, software and publications. It's an electronic repository of OS/2 software development tools and demos available at no cost. And it's more.

Early Code Programs let you test new OS/2 versions with your product. Migration Workshops provide software and technical assistance to port applications to OS/2. But the Developer Assistance Program doesn't end with the completion of your software program.

Membership entitles you to free listings on our OS/2 CD-ROM Application Demonstration System and application directory. Take advantage of direct marketing offerings at reduced prices through the OS/2 Direct Marketing Center. And there are many more offerings, including business shows, targeted mailings, in-package promotions and instructional videos.

To find out more about the DAP and how to qualify for specific programs, enter "GO OS2DAP" on CompuServe, or call **1 407 982-6408.** You'll see we've developed a powerful program to help develop yours.



Our CD-ROM Professional Developers' Kit provides beta-level code for operating system and tools and includes online technical reference libraries.

IBM®

Circle No. 807 on Inquiry Card

Chris Corry

Preview ++

Pity the OS/2 programmer. For the last couple of years, OS/2 has been treated like the Rodney Dangerfield of the software industry, and the phenomenal success of Windows has only made things more frustrating for OS/2 developers. With the majority of resources being poured into DOS, Windows, and Windows NT, OS/2 programmers have traditionally been saddled with a host of lackluster development tools and compilers.

Since the introduction of OS/2 2.0, things have started to change, and now I am happy to say that the OS/2 developer's proverbial ship is about to come in. Shortly, Borland and IBM will be unleashing their new C++ compilers, and OS/2 application development will never be the same. These products promise to deliver the same high-end features and performance currently only available under DOS and Windows.

As of January 1993, both products discussed here were in beta test and not expected to ship for several months. Products in beta test are typically dynamic creatures whose feature sets or components are subject to alteration (or even radical changes) at any time preceding the ship date. It would appear that these compilers are pretty far along, and the odds of any major changes being made to either product are slim.

Borland C++ for OS/2

Fans of Borland's C++ for Windows will immediately recognize the OS/2 version as a direct descendent. The product is amazingly complete, considering that every aspect of the system was written from scratch to take advantage of OS/2 2.0's 32-bit architecture and advanced operating system services. The base package includes an integrated development environment (IDE), a Presentation Manager (PM) hosted version of Turbo Debugger, the Resource Workshop, a command-line compiler, linker, and a host of smaller utilities and tools.

The compiler (which is the same for the command-line version and IDE) allows programmers to build full-fledged 32-bit programs that operate in text mode or as native PM applications. Unlike the IBM product, Borland does not support the development of virtual device drivers.

The Borland compiler includes support for standard ANSI C, and its C++ implementation is based on the AT&T cfront 2.1 specification. Borland provides full support for templates, a C++ construct allowing for the creation of generic classes that manipulate anonymous data. However, exception handling, an important feature just starting to show up in C++ compilers, is not supported. Precompiled headers are available in the command-line and IDE versions of the compiler.

As you would expect, the package supports programming of OS/2's advanced features, including multithreading and dynamic link libraries. The full PM API set is covered, including the new extensions added to support the Workplace Shell. Borland also includes a set of IBM developed precompilers and utilities that allow you to build system object model (SOM) applications. SOM is a powerful, new, object-oriented programming mechanism that allows for language-independent development, extension, and use of classes between compilers.

The IDE that comes with Borland C++ for OS/2 is an impressive

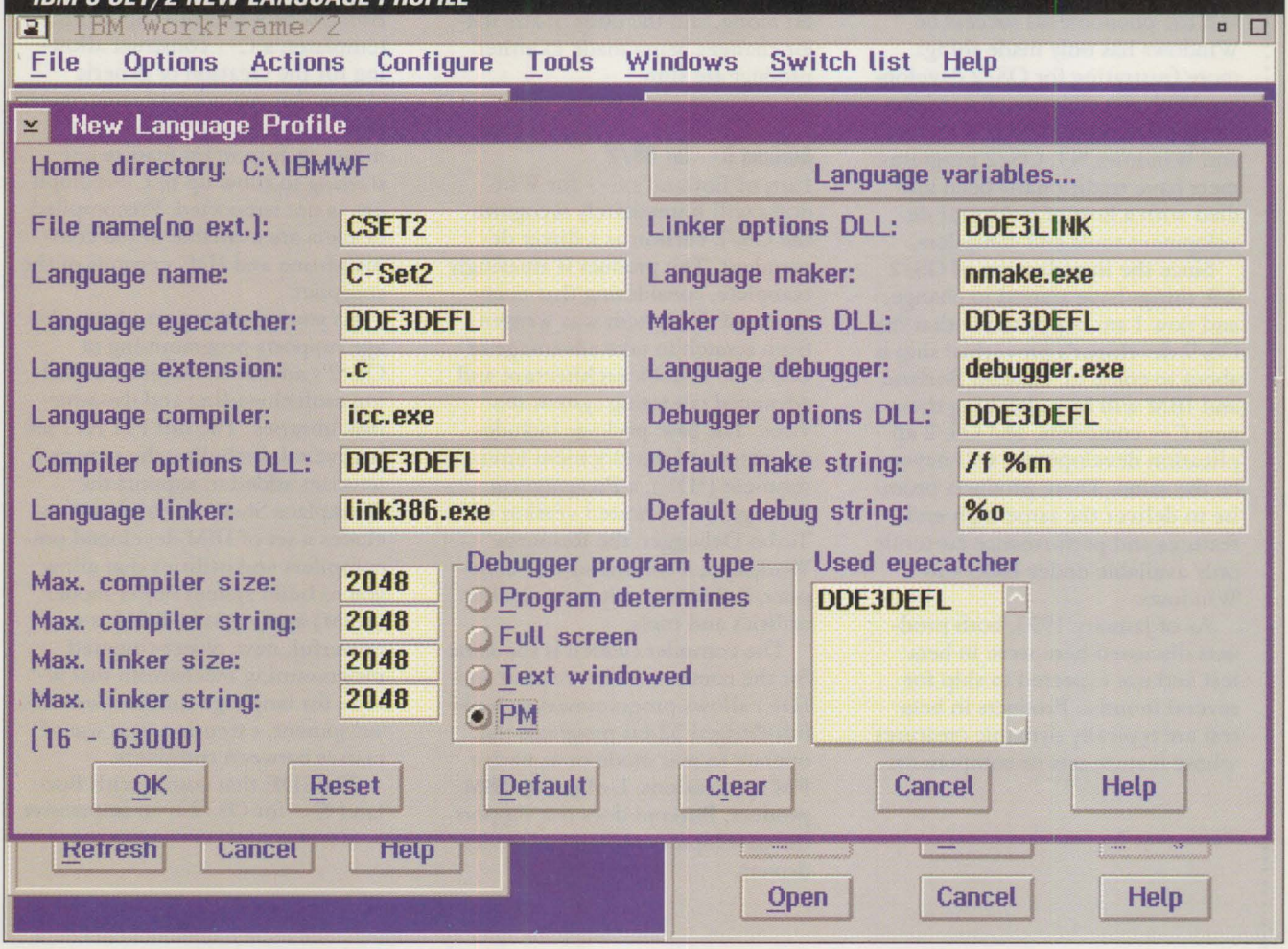
programming environment and a faithful port of the Windows product. A casual survey of the IDE reveals the obligatory toolbar (Borland calls them SpeedBars), project and edit windows, and an integrated debugger. The OS/2 IDE also includes a number of important features that differentiate it from its Windows-based sibling. Because OS/2 is a multitasking operating system, the OS/2 IDE allows developers to start tasks and let them execute in the background. It is a tremendously convenient feature, and users will rapidly become accustomed to kicking off background operations and immediately moving on to the next task.

The IDE extensively uses the Workplace Shell convention of displaying pop-up menus when the right mouse button is clicked over a window. This ability provides the user interface with an exceptional amount of flexibility. For example, to save the current file, you can select the *Save* option from the main file menu, use the appropriate accelerator key combination, click on the toolbar, or bring up the pop-up menu for the selected edit window.

The integrated editor is standard Borland fare. In addition to the standard Borland key bindings, Borland provides alternative keyboard mappings that provide Brief, Epsilon, or CUA behaviors. A keyboard definition compiler is also provided that allows you to build custom keyboard layouts. In practice, however, creating an emulation of your favorite editor can be a trying experience that requires perseverance and a good understanding of the IDE's macro language.

Syntax highlighting allows certain elements of your source code to be displayed in different colors while editing. For example, you can set variable names to appear in blue while displaying reserved words in red. Displaying your code in this manner lets you catch typos and simple mistakes at a much earlier stage of the development cycle—but be prepared for an adjustment period. I definitely found the multicolored source window to be an acquired taste.

IBM C SET/2 NEW LANGUAGE PROFILE



Like its siblings, the IDE completely does away with *makefiles*. Instead, the programmer creates projects that seamlessly automate the generation of internal dependency lists. Because the IDE takes care of these things under the covers, any changes made to source code will only force the recompilation of those modules that need it.

I was surprised to find that the excellent class browser built into the Windows IDE is absent from the first version of the OS/2 product. A browser allows the programmer to inspect individual class characteristics and call up the source code for specific class methods by clicking on a hierarchy diagram. Although Borland has indicated that the browser will make an appearance in a future version, its exclusion from the first release is disappointing.

The version of Turbo Debugger that ships with the compiler looks unlike any other debugger from Borland. It is a fully graphical application that runs as a native PM program. Like the IDE, it makes judicious use of toolbars and pop-up menus. The debugger allows large, multithreaded programs to be tracked from the central window. Threads can be individually traced, suspended, and resumed. Breakpoints and watchpoints are easily set. The debugger also allows you to stop execution after a PM program has received a particular message in its message queue. OS/2 exceptions can be passed onto the program or trapped so that the state of the program can be evaluated in a post-mortem fashion.

Debugging programs with TD is simple and almost enjoyable. It is, however, very easy to find yourself with many windows open at the same time. When you add your program's windows to the equation, the screen clutter can be annoying on a high-resolution screen and downright obnoxious at standard VGA resolutions. I suppose if you want to look at a dozen different things at the same time, you better be prepared to have a dozen windows open on your screen. Regardless, a higher-resolution screen is definitely recommended.

Borland C++ for OS/2, despite all of its similarities to the Windows version, is lacking several key components that have helped to make the other professional Borland C++ products so successful. Notably absent are the Object Windows Library (OWL) and Turbo Vision applications frameworks. Because Turbo Vision is a collection of classes that only support the creation of full-screen text mode user interfaces, this omission should cause little concern.

The lack of support for OWL, however, is a little more significant. The OWL libraries for Windows are a collection of user interface classes that provide comprehensive support for many of the Windows APIs. OWL has proven to be very popular, and many developers use the libraries exclusively for all GUI programming. Programmers who have a large amount of OWL-based code will have to wait until the OWL libraries are ported to OS/2 before they can migrate their applications. Borland is developing OWL libraries for OS/2, but there is no word yet on when they will be available.

Although none of the Borland applications frameworks is supplied with the product, the strong Borland container class libraries are included. The classes come in template form, with a set of explicit classes included to provide backward compatibility with previous versions of the library. Included in the library are classes to implement lists, stacks, queues, dictionaries, binary trees, and iterators, to name just a few.

IBM Corp. C Set/2 v. 2.0

When IBM first introduced OS/2 2.0, it shipped a comprehensive development environment that centered on the C Set/2 compiler. This sophisticated compiler was subsequently used by virtually every OS/2 developer, not only because it was the only 32-bit OS/2 compiler shipping but because it also proved to be a competent performer. Any compiler vendor that wants to challenge IBM will have to penetrate this significant installed base, and IBM is not sitting on its laurels. IBM's languages division in Toronto, Ont., Canada, is now preparing the next release of C Set/2 that, as it turns out, is a very capable C++ compiler. The new version of the product includes the command-line compiler, linker, updated graphical debugger, class browser, profiler, and a number of smaller support programs and libraries.

Borland has always been in the position, at least when its primary competitor has been Microsoft, of having the most robust and complete C++ implementation on the market. IBM, however, has leapfrogged Borland by providing full exception handling in addition to ANSI X3J16 (up to the June 5, 1992 working draft) compliance. Exception handling provides a tremendously powerful way to capture and process abnormal program events. Unlike some other implementations of exception handling that rely on a simple *setjmp/longjmp* mechanism, the IBM exception-handling support reliably ensures that class destructors are called when an exception forces an object instance out of scope.

The compiler, not surprisingly, resembles the original IBM C Set/2 compiler in both form and function. In addition to the significant C++ implementation, however, a few welcome additions have

been made to the compiler since the initial release. For example, all run-time library functions are now re-entrant, which means that functions such as *sprintf* can now be safely used in multithreaded programs. Previously, developers working on multithreaded applications had to shy away from using particular library calls or risk encountering some very frustrating debugging sessions.

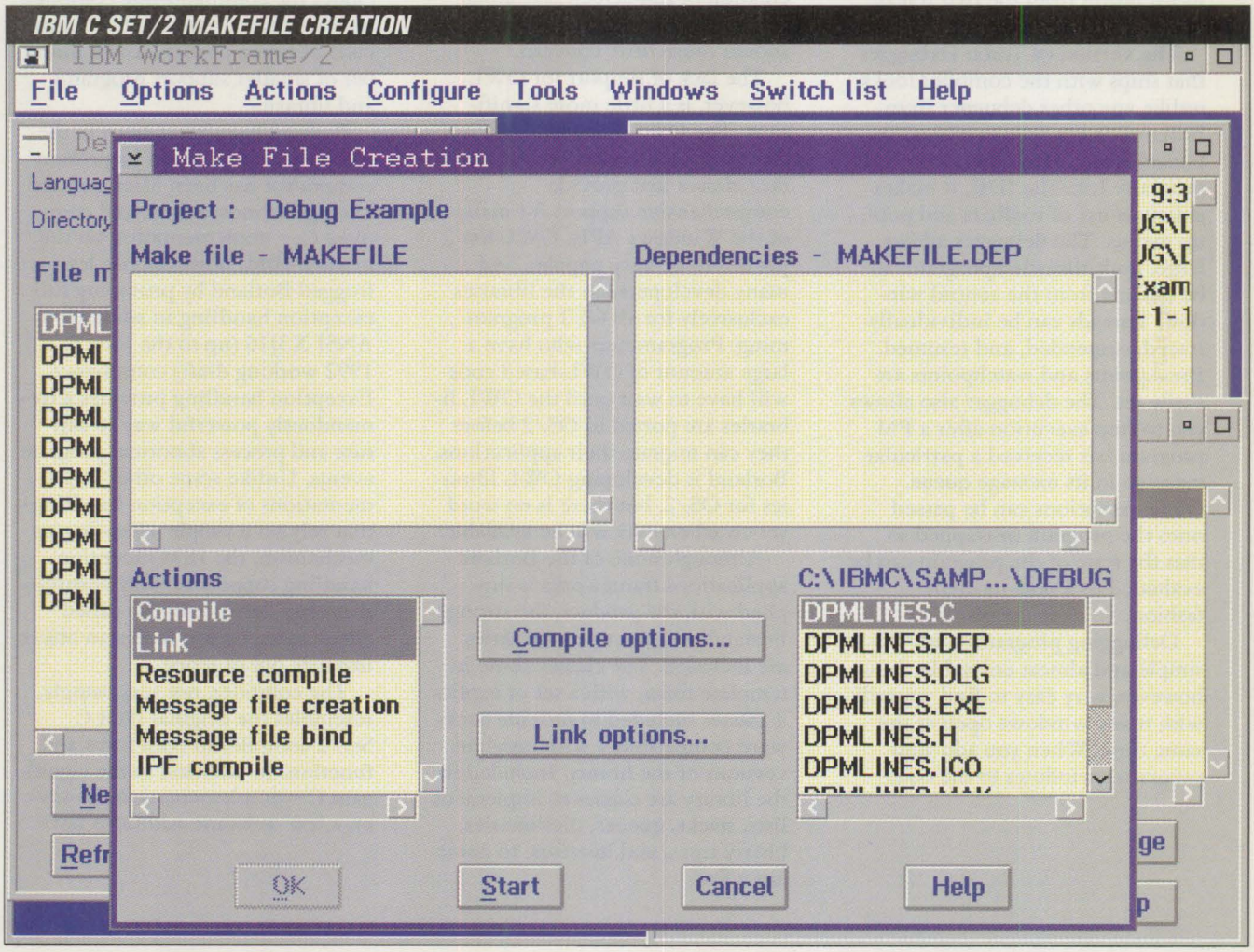
There are, also, a number of new compiler options. One intriguing compiler switch optimizes your code to run on Intel's forthcoming Pentium (586) chip. Another controls the use of precompiled headers. The IBM compiler also provides excellent floating-point support in the form of IEEE 80-bit long double types.

If you want to develop PM programs with this C++ compiler, you will need to purchase the IBM OS/2 Developer's Toolkit, which is available separately. Support for SOM, however, is supplied with the compiler and requires no additional components (although the creation of SOM objects that use PM will require the toolkit).

IBM's IDE design strategy diverges sharply from Borland's. While Borland has chosen to build its compiler directly into its IDE and bind its development tools into a single environment, IBM has chosen the completely opposite approach. The WorkFrame/2 product, sold separately from the compiler, is a general-purpose IDE. Out of the box, it does almost nothing. Instead, it provides a generic development environment framework into which the developer can plug virtually any third-party tool. This flexibility allows the WorkFrame to work with almost any compiler (even compilers designed for DOS or Windows), albeit not quite as smoothly as the Borland IDE.

One of the advantages to this strategy is that developers can use the editor of their choice. With the Borland IDE, however, you have little choice but to use the built-in editor or spend the time creating a keyboard map that emulates your

IBM C SET/2 MAKEFILE CREATION



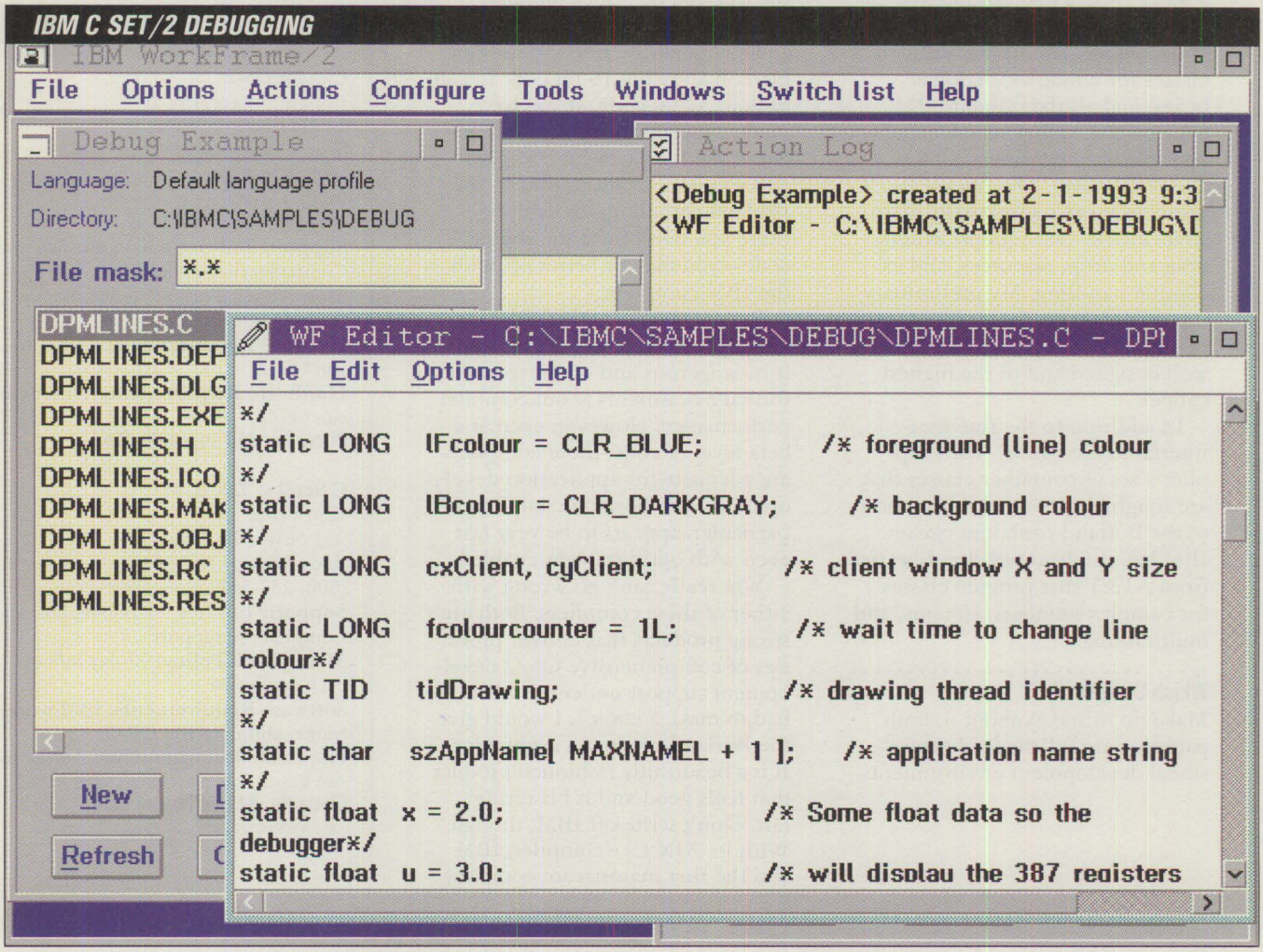
editor. Because most professional developers have an almost fanatical allegiance to a particular editor, I am sure that many will prefer the WorkFrame approach over Borland's. Also, Borland provides the necessary dynamic link libraries to integrate its compiler into the WorkFrame.

The downside to the WorkFrame's philosophy is that even the most WorkFrame-aware compiler and tools don't feel as closely coupled as the tools in the Borland IDE. Depending on the capabilities or limitations of your editor, for example, you may not be able to click on error messages and go to the appropriate line in your source code. Even WorkFrame *makefile* creation, which in many ways mimics the Borland environment, is more difficult and feels a little less polished.

Unfortunately, neither the WorkFrame nor Borland IDE is seamlessly integrated into the Workplace Shell. Despite all their respective strengths, they still look and feel more like OS/2 1.x applications. IBM has hinted that a Workplace-Shell-oriented version of the WorkFrame is under development, and developers at Borland have commented that more complete Workplace Shell integration is being considered for the next product release.

Borland's Turbo Debugger resembles IBM's IPMD debugger in many ways, and it is probably safe to assume that the original IPMD tool served as an inspiration for the Borland development team. This version has been significantly overhauled with a number of new features, the most significant, of course, being the support for programs written in C++. The new release also adds convenient toolbar-like buttons to source windows (although the icons used in my beta version made it difficult to decipher what each button did) and replaces the traditional code windows with a notebook control that lets you easily flip between different source modules.

Probably one of the nicest features of IPMD is its monitor win-



dow that displays the value of various tokens as you inspect them. Aggregate constructs (classes, structures, and unions) that contain other elements can be collapsed or expanded depending on the level of detail you are interested in seeing. The user interface is not as polished or usable as Borland's Turbo Debugger, and there are gaps in IPMD's capabilities (setting automatic breakpoints on PM messages, for instance). Still, the IPMD debugger is a good tool that should be more than adequate for anything but the most demanding debugging tasks.

One of the most exciting aspects of the IBM compiler is the comprehensive set of PM class libraries that come with the product. The IBM user-interface library provides a set of classes that abstract the entire messaging model that PM operates under. The library delivers classes for windows, dialogue boxes, and all the standard PM controls (including new OS/2 2.0 controls like containers, sliders, and notebooks). The library also provides more conceptually abstract classes that support events, drag and drop, resources, and exception handling. My preliminary experiences with these classes indicate that the library is exceptionally well designed and of the highest caliber.

In addition to the fine user-interface class library, IBM supplies a set of container classes that are roughly equivalent in function to the Borland container classes. IBM has also licensed class libraries from AT&T that provide classes for complex numbers, streams, and multitasking.

Which Compiler?

Make no mistakes about it, both packages are full-fledged professional development environments. If you are looking for a cheap, hobbyist compiler, I suggest you look elsewhere. If, on the other hand, you are a professional developer who depends on programming OS/2 for your bread and butter, you will want to invest in one of these compilers. You should also take a close look at these offerings if you are responsible for purchasing tools for a corporate development shop or in-house programming department.

So which of these systems promises to be the stronger product? One thing to take into account when making your decision is compatibility. The IBM compiler fully supports interaction with older 16-bit code compiled for OS/2 1.x, including 16-bit functions calling back to 32-bit functions. Although the Borland compiler supports similar interactions with 16-bit code, at deadline Borland did not plan on allowing 16-bit code to call back to 32-bit code. If you have made a substantial investment in C Set/2 v. 1.0 or a 16-bit compiler and you don't want to port to the Borland compiler—or can't because you don't have the source code—you may be better off with the IBM compiler.

Because both of these products are only available in beta versions, it is dangerous and unfair to make formal assessments about compiler performance. However, even at a beta level of code, these compilers are adequate for application development. The Borland compiler, in particular, appears to be very fast even with optimizations enabled.

You really can't go wrong with either of these compilers. Both are strong products that deliver promises of comprehensive OS/2 development support under C++. If I had to make a choice, I would give the Borland product a slight edge. It is a beautifully fashioned product that feels good and is blistering fast. Don't write off IBM, though. With its AIX C++ compiler, IBM was the first mainstream company to ship a compiler that fully supports C++ exception handling. Now this capability will be available to the PC programmer. And, although the container classes that come with each of the compilers seem equal in scope, the IBM compiler wins the class library crown with its comprehensive PM user interface classes. It looks like OS/2 programmers are about to be faced with a no-lose proposition—finally. ■

Chris Corry is a software developer working for American Management Systems of Arlington, VA. His expertise includes object-oriented programming, GUIs, imaging, and workflow technologies. He holds a Bachelor's degree from the University of Pennsylvania. He can be reached through CompuServe @ 70421,2367.

The Nitty Gritty

C++ for OS/2
Borland International
1800 Green Hills Rd.
Scotts Valley, Calif. 95067
(408) 438-8400
(800) 331-0877
Fax: (408) 439-9343
Support: Toll-free telephone, fax, and CompuServe support
Moneyback Policy: 90-day money-back guarantee
Software Requirements: Ability to run OS/2
Hardware Requirements: Ability to run OS/2
Circle No. 840 on Inquiry Card

C Set/2 v. 2.0
IBM Corp.
951 N.W. 51st Street
Boca Raton, Fla. 33431
(800) 237-5511
Support: Toll-free telephone, bbs, and CompuServe support
Moneyback Policy: 90-day money-back guarantee
Software Requirements: IBM workframe, ability to run OS/2
Hardware Requirements: Ability to run OS/2
Circle No. 841 on Inquiry Card.

At Your Service!

THE PROGRAMMING CUTTING EDGE

Free Issue!

COMPUTER LANGUAGE Free Issue Offer!

☐ **Yes! Send me a FREE issue of *Computer Language* and start my one year subscription (12 issues) for just $29.95. I save 36% off the cover price! Please bill me.**

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

4OS2

Single-copy price is $3.95. Please allow up to six weeks for delivery of first issue. Canada/Mexico add $6 per year for surface mail. Overseas add $15 per year for surface mail or $40 per year for air mail. GST # 124513185.

COMPUTER LANGUAGE

STAY ON-LINE WITH 32-BIT POWER!

Great Deal!

IBM OS/2 DEVELOPER Subscription Offer!

☐ **Yes! I want to subscribe to *IBM OS/2 Developer*. Send me a one-year (4 issues) subscription for just $39.95! Please bill me.**

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

4OS2

Delivery of first issue is within 8 weeks. Canadian/Mexican and international surface orders, add $16 per year for postage. International airmail, add $30 per year. Price includes GST.

IBM OS/2 Developer

FREE PRODUCT INFORMATION

Free Info!

COMPUTER LANGUAGE-OS/2 Supplement Reader Service Card

Send in this card for FREE product information from *Computer Language-OS/2* Supplement advertisers!

☐ Ms. ☐ Mr. (please print) _____

Company _____

Address _____

Dept./Mail Stop _____ Phone ( ) _____

City _____ State _____ Zip _____

Circle the Reader Service Numbers of the products you'd like to know more about. This offer expires July 31, 1993.

801	806	811	816	821	826	831	836	841	846
802	807	812	817	822	827	832	837	842	847
803	808	813	818	823	828	833	838	843	848
804	809	814	819	824	829	834	839	844	849
805	810	815	820	825	830	835	840	845	850

Recycled Paper

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS MAIL PERMIT NO. 1286 BOULDER, CO

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTER
LANGUAGE
Miller Freeman Publications
P.O. Box 51258
Boulder, CO 80321-1258

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS MAIL PERMIT NO. 1099 SKOKIE, IL

POSTAGE WILL BE PAID BY ADDRESSEE

IBM OS/2 Developer
P.O. Box 1079
Skokie, IL 60076-9772

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS MAIL PERMIT NO. 786 PITTSFIELD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTER LANGUAGE
OS/2 Supplement
Reader Service Department
P.O. Box 5358
Pittsfield, MA 01203-9890

12 ISSUES OF
COMPUTER LANUGAGE
FOR ONLY $29.95!

4 ISSUES OF
IBM OS/2 DEVELOPER
FOR JUST $39.95!

FREE PRODUCT INFORMATION
FROM COMPUTER LANGUAGE
ADVERTISERS!

See flip side!


And the winner is...
In Recognition of your contribution to a new class of OS/2 2.0 APPLICATIONS
IBM CORPORATION
presents an award to
Intelligent Environments
for shipping
Applications Manager
AM
BEST OS/2 EXPLOITATION
BEST PRODUCTIVITY GAINS
BEST DATABASE INTEGRATION
BEST CUSTOMER SATISFACTION
BEST ADVANCED CLIENT/SERVER TOOL
APPLICATIONS MANAGER • INTELLIGENT ENVIRONMENTS •
aM
Intelligent Environments
USA 508•640•1080 ◆ EUROPE +44•932•772266
•BOSTON•CHICAGO•HOUSTON•LOS ANGELES•LONDON•FRANKFURT•STOCKHOLM•
INTELLIGENT ENVIRONMENTS INC., 2 HIGHWOOD DRIVE, TEWKSBURY, MA ALL PRODUCTS AND PRODUCT NAMES MENTIONED ARE REGISTERED TRADEMARKS OF THEIR RESPECTIVE COMPANIES.
Circle No. 808 on Inquiry Card

Let Gpf write the GUI you design

Gpf

SYSTEMS, INC.

Using the powerful point and click visual programming environment of Gpf*, you can prototype, test and generate a complete OS/2 PM GUI in a few hours or days rather than the weeks or months required to hand code the same design. Even a relatively simple GUI can require writing thousands of lines of code, but with Gpf you simply draw your user interface on the screen. The integrated dialogue editor of Gpf permits actions and context sensitive help to be linked to controls as you create them. Gpf then generates error free ANSI C, complete with embedded SQL statements.

Gpf is optimized to take full advantage of OS/2 PM, the most powerful and robust GUI system available. Since Gpf code directly accesses the PM API, there is no run time module to distribute with your application and no added overhead or royalties.

Gpf keeps the entire design definition in one file. This means single point maintenance for easy updating and archiving. From this file, Gpf generates the C source file as well as .H, .RC, .IPF, DEF, .IDS, .MAK, etc.

Gpf 2.0 Supports:

- 32 Bit Code Generation and CUA '91 Controls.
- Simple and direct linkage of the interface to program logic, built in or user defined functions.
- Direct association of help screens with controls and complete integration into the PM Help Presentation Facility.
- Flexible use of Presentation objects (fonts, colors, etc.) with controls and windows (client area and frame).
- Simple inclusion of bitmaps for use on About screens, user-defined buttons, and menu or pulldown entries.
- Automatic embedded SQL statements to read OS/2 Database Manager tables directly into combo or list boxes.
- Multi-thread programming.
- Multiple source file generation.
- Automatic creation of controls that scale with window size.
- Inclusion of user defined controls.

Also Available
GpfTools
• Reusable Objects
• Auto Documentation
• Browsing

Try us out.

Order: Gpf for just $995.^oo and GpfTools for $195.^oo or SPECIAL both for $1,140.^oo (A $50.^oo savings)

Call **Gpf Systems Inc.** at:
(203) 873-3300 or (800) 831-0017 - fax (203) 873-3302.
Free demo software available
30 Falls Rd., P.O. Box 414, Moodus, CT 06469

Information on International Distributors available – call Gpf Systems Inc.

IBM Business Partner

* GUI Programming Facility

Circle No. 809 on Inquiry Card